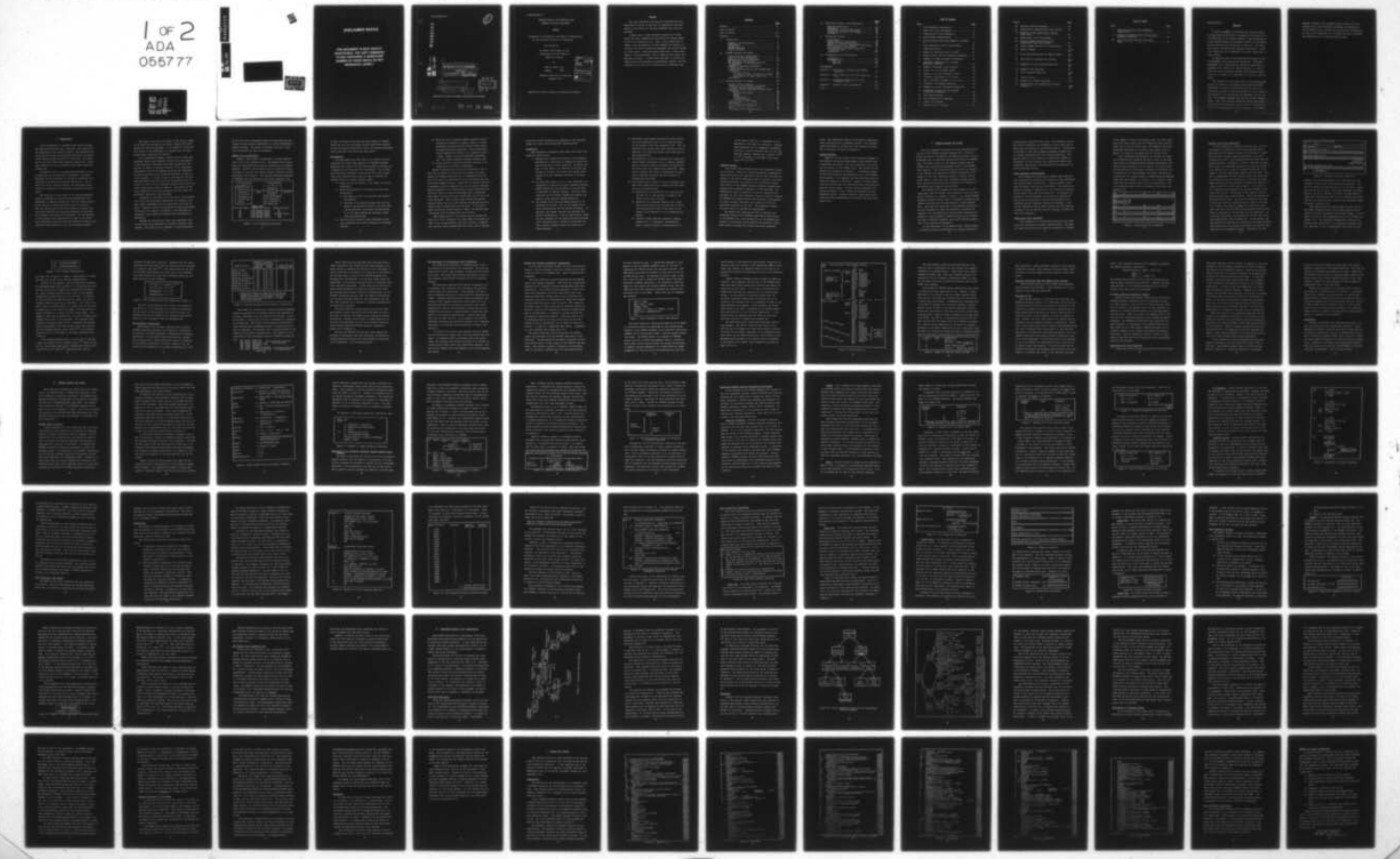
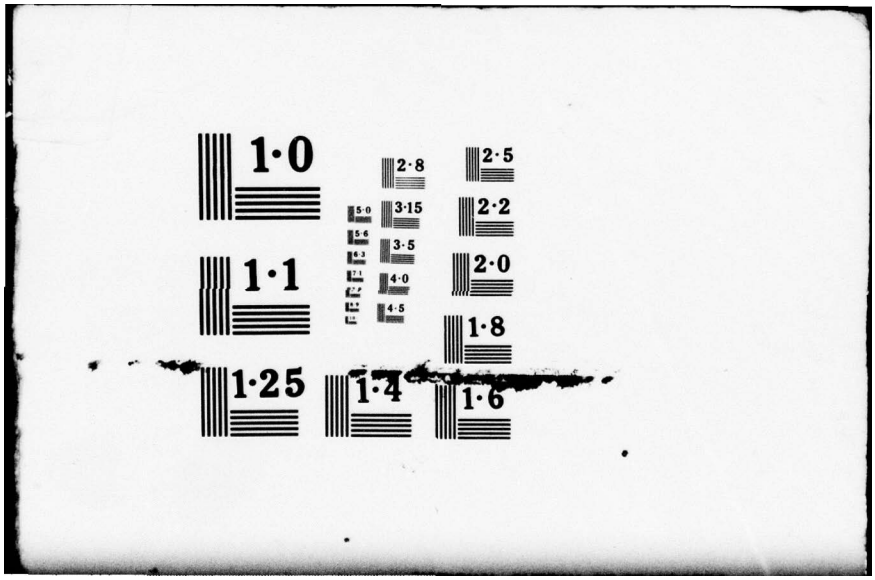
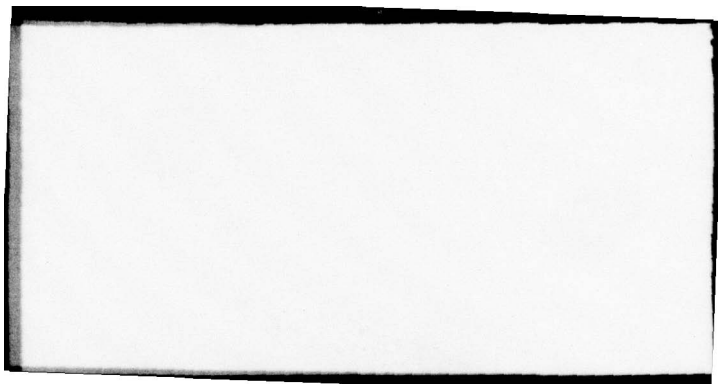
AD-A055 777 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
SOFTWARE TOOL(S) FOR EVALUATING THE EFFECTS OF FINITE WORDLENGT--ETC(U)
DEC 77 G A KLEIN

UNCLASSIFIED AFIT/GCS/EE/77-6 NL

1 OF 2
ADA
055777




1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0



DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

AFIT/GCS/EE/77-6

AD A055777

AD No. _____
DDC FILE COPY

(6) SOFTWARE TOOL(S) FOR EVALUATING THE EFFECTS OF FINITE WORDLENGTH.

THESIS (10)

(14) AFIT/GCS/EE/77-6

(10) Gary A. Klein
Captain USAF

(9) Master's thesis,

(11) Dec 77

(12) 185p.

DDC
RECEIVED
JUN 20 1978
E

Approved for public release; distribution unlimited.

Ø12 225

AFIT/GCS/EE/77-6

SOFTWARE TOOL(S) FOR EVALUATING THE EFFECTS OF FINITE WORDLENGTH

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Gary A. Klein, B.S.
Captain USAF

Graduate Electrical Engineering
December 1977

Preface

The n-bit simulation tool has been developed with the hopes that it can be of some help to engineers in the evaluation of the effects of varying wordlength upon their algorithms.

I would like to thank Professor Richard and Mr Stan Musick for their comments and help during the design phases of the n-bit simulation tool. Also, I owe a great deal of thanks to my two advisors, Dr Peter Maybeck and Captain J B Peterson, for their on-going encouragement and advice throughout this period. They were always there when I needed them. Lastly, I would especially like to thank my wife, Karen, who was also my typist. I would never have made it without you. Thank you for your patience, understanding, support, and your endless flow of love, not to mention your outstanding work.

Contents

List Of Figures

List Of Tables

AFIT/GCS/EE/77-6

Abstract

In the development of estimation and control systems, the algorithms developed are, in general, very sensitive to the wordlength of the onboard computer. The representation of the desired design can be greatly affected by the quantization resulting from truncation or round off. The result can be even more severe than numerical imprecision: numerical instability can be generated that render algorithms totally useless.

An important part of the algorithm development is the determination of the appropriate wordlength. The proper wordlength can be determined by running the algorithm on a simulated n-bit machine for several values of n. The results can then be evaluated against the performance specification to determine the wordlength and precision requirements.

The objective of this thesis was to process algorithms written in FORTRAN on the Control Data Corporation (CDC) 6600/CYBER 74 computer systems such that the results obtained were similar to those obtainable on an n-bit machine. The problem of n-bit simulation was addressed from two levels: the assembly language level and the FORTRAN language level. Each level involved designing a preprocessor which would modify the algorithm's code so that the numerical effects of n-bit wordlength could be realized. The assembly

(cont fr p viii)

language (COMPASS) level approach failed, however, an n-bit simulation tool was successfully developed and implemented at the FORTRAN level. Several user options were incorporated into the n-bit simulation tool to enable the user to simulate the characteristics of various computer types.

I. Introduction

In the development of estimation and control systems, the algorithms developed are, in general, very sensitive to the wordlength of the onboard computer. The representation of the desired design can be greatly affected by the quantization resulting from truncation or round off. The result can be even more severe than numerical imprecision: numerical instabilities can be generated that render algorithms totally useless.

An important part of the algorithm development is the determination of the appropriate wordlength. The proper wordlength can be determined by running the algorithm on a simulated n-bit machine for several values of n. The results can then be evaluated against the performance specification to determine the wordlength and precision requirements.

The objective of this thesis is to process algorithms written in FORTRAN on the Control Data Corporation (CDC) 6600/CYBER 74 computer systems such that the results obtained are similar to those obtainable on an n-bit machine. Then the results would reflect the degree of accuracy that would have resulted from actually processing the algorithm on a computer with that n-bit wordlength. For the purposes of this report, n-bit simulation will refer to the effect n-bit wordlength would have upon the numerical accuracy of an algorithm being processed on an n-bit wordlength machine.

This type of tool would be useful in the initial stages of the selection process for an onboard computer that would process the desired algorithms. It is essential to know the full impact of the wordlength of a computer upon the algorithms before selecting the computer.

For a hypothetical example, suppose the Air Force just purchased several mini-computers having 16-bit wordlengths. It was thereafter discovered that these computers did not furnish the degree of accuracy required by the algorithms. Considerable time may be expended rewriting the algorithms to process properly on the purchased computers, if that is at all possible. In fact, it may be the case that the minimum wordlength which could meet the requirements would be 24-bits or that a 16-bit wordlength computer with double precision capabilities would be required. Application of the n-bit simulation tool before purchase could have saved the government significant time and money.

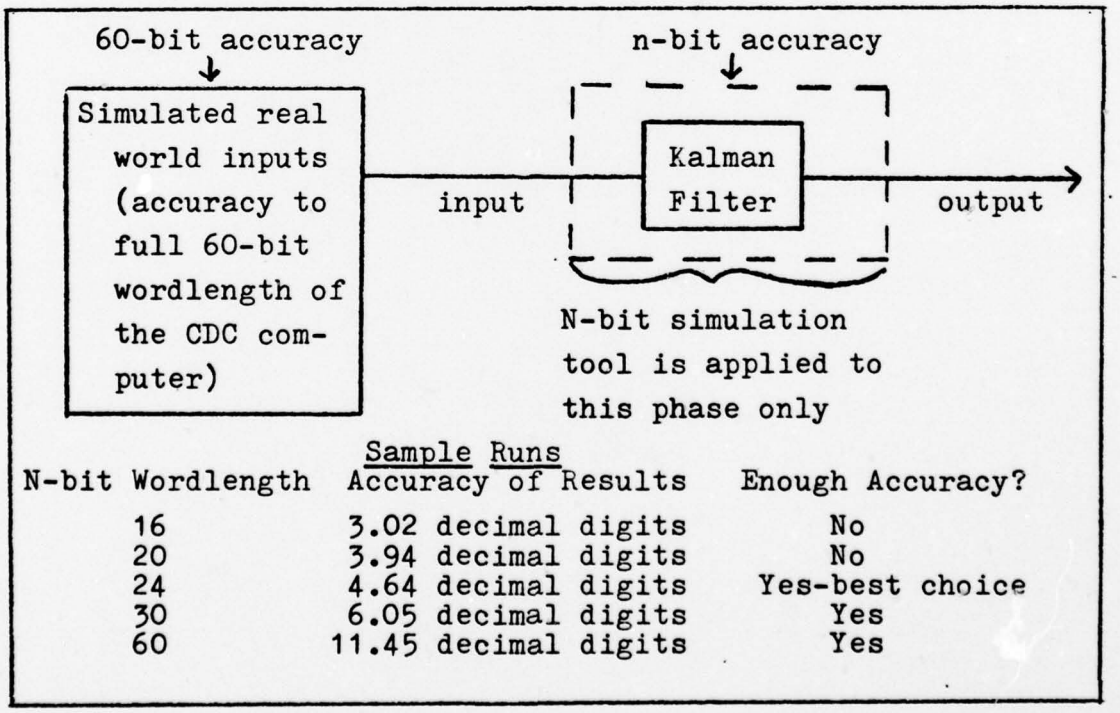
Similarly, at the other end of the scale, this software tool could save government expenditures for computers with much longer wordlengths than the algorithms could ever require. Computer costs go up rapidly with longer wordlengths. Tailoring of the computer wordlength to the algorithm requirements could therefore eliminate some unnecessary expenditures.

Through the utilization of the n-bit simulation tool, an algorithm can be processed with several different wordlengths. The results can be compared to results generated

by the full 60-bit wordlength (which is longer than any foreseeable onboard computer wordlength) of the CDC 6600/CYBER 74 computer systems. The degree of accuracy lost due to each n-bit wordlength can then be determined.

Example of Its Application

Figure 1 can be used to demonstrate a typical application of the n-bit simulation tool. The Kalman Filter represents the algorithms which would be processed by the onboard computer. However, this procedure did not correctly reflect the degree of accuracy the onboard computer (having a shorter wordlength) would achieve. The original algorithm (programed in FORTRAN) would be modified by the n-bit simulation tool



Sample Runs

N-bit Wordlength	Accuracy of Results	Enough Accuracy?
16	3.02 decimal digits	No
20	3.94 decimal digits	No
24	4.64 decimal digits	Yes-best choice
30	6.05 decimal digits	Yes
60	11.45 decimal digits	Yes

Figure 1 N-bit Simulator Application

so that its results would show the exact amount of significance that an n-bit wordlength computer would have produced. It may be executed several more times with varying wordlengths until sufficient information has been gained.

Requirements

The requirement for this thesis is to develop an n-bit simulation tool which when applied to a FORTRAN programed algorithm will produce the near exact arithmetic accuracy as a computer of the specified n-bit wordlength. Five options were considered necessary to provide the user with a versatile n-bit simulator which could be used to simulate the characteristics of various computer types:

1) Allow the user to specify n, the number of bits per wordlength,
2) Permit specification of floating point word characteristics:
    a) The number of bits to be used for the exponent and mantissa
    b) Position of the implied decimal point with respect to the mantissa, either to its left (making the mantissa a fractional representation) or to its right (making the mantissa a fixed point representation),
3) Give the user an option to have arithmetic operations performed with either truncation or rounding effects,

4) Allow the user to specify whether arithmetic calculations are performed in single, double, or triple wordlength precision with the results being stored as single precision quantities. This option essentially simulates the effects of a computer having a single, double, or triple wordlength accumulator but only single precision storage variables, and

5) Allow the user to specify different portions of an algorithm for different wordlengths.

Another characteristic of the n-bit simulation tool is that when an overflow is detected for a given n-bit wordlength, the maximum value that can be represented with the n-bit word should be substituted for the overflowed word and an overflow message printed. Overflows are treated differently depending upon the computer type, but it was felt maximum number replacement for overflows would better demonstrate the numerical effects of shorter wordlength upon an algorithm than simply truncating the most significant bits of the value that overflowed. When a word's value exceeds the maximum positive number the word could contain, the maximum positive number would be substituted into the word. If the word's value exceeds the maximum negative number, the maximum negative number would be substituted into the word.

Some general goals of this thesis were to provide the user with a n-bit simulation tool that would be reliable, be easy to use, be maintainable, provide an option to communicate overflow error messages when they occur, and to employee

structured design techniques in programing so the resulting design can attain certain desirable characteristics.

Assumptions

Some fundamental assumptions upon which this thesis was based are listed below:

1) Applying n-bit simulation effects to all arithmetic expressions is sufficient to simulate an entire program effectively in n-bit precision. The major purpose of this thesis is to provide a tool which can be used to evaluate the effects that varying wordlengths have upon arithmetic accuracy of an algorithm.
2) The algorithms which are to be n-bit simulated, are programable in American National Standards Institute (ANSI) FORTRAN and are executable on the CDC 6600 or CDC CYBER 74 computer systems. Potential users currently use the FORTRAN language and execute programs on these computer systems regularly.
3) The maximum fixed point wordlength desired to be simulated is 48 bits. This is the largest fixed point representation the CDC system can handle. In addition, the largest exponent and mantissa that would be required are 11 and 48 respectively. These are also maximums for CDC single precision floating point representations. Within the forseeable future, onboard computers would not exceed any of these maximums.

4) Additional core storage and execution time required for n-bit simulation will not be a limiting factor in the design of the n-bit simulation tool. This is partially true because this n-bit simulation tool is expected to be used only several times each year by any projected user.

5) The quantities to be n-bit simulated are single precision numeric quantities (i.e. no logical, alphanumeric, or double precision variables and quantities are used within the program being simulated). Again, this is because this thesis is addressing the problem of numerical precision and the stability of mathematical algorithms.

6) All floating point values to be n-bit simulated will have three characteristics in common with CDC floating point representations:

   a) Mantissa will be normalized (i.e. the most significant bit of the mantissa is always a one, except in the case of zero).

   b) Exponents will represent powers of two, which would be multiplied with the mantissa to obtain the floating point value of the word, for instance, not by powers of 16 for hexadecimal machines.

   c) Computers being simulated represent negative numbers with 1's complement representation, where a negative number is represented by a 1

in the sign bit and the 1's complement of its magnitude in the other bit positions. Floating point exponents for 2's complement representation have a positive exponent range of one less than the negative exponent (for instance, 2^{-128} to 2^{+127}, whereas 1's complement would range from 2^{-127} to 2^{+127}).

General Approach

This thesis study addressed the n-bit simulation problem at two levels. Each level involved designing a preprocessor which would modify the algorithm's code so that the numerical effects of the n-bit wordlength could be realized. The first approach attempted to accomplish n-bit simulation through modification of the COMPASS assembly language representation produced from a FORTRAN compilation of the algorithm. The COMPASS code would be modified by an n-bit simulation preprocessor so that every arithmetic operation would be replaced by COMPASS code sequence which, when executed, would generate the exact accuracy attainable on a computer of the n-bit wordlength specified by the user. This approach, however, was unsuccessful in meeting the requirements.

The second level approach was able to perform n-bit wordlength affects upon a FORTRAN algorithm by modification of the FORTRAN code. This modification of the FORTRAN code requires a preprocessor which analyses the arithmetic statements within a program and replaces them with subroutine

calls. The subroutines perform the arithmetic operations while simulating n-bit wordlength affects. This approach proved successful in solving the n-bit simulation problem and is discussed in detail in Chapter 3.

Chapter Synopsis

Chapter 2 of this thesis report contains a summary of the analysis and design accomplished at the COMPASS level. A successful prototype of the n-bit simulation tool could not be achieved at the COMPASS level. The reasons for the failure are elaborated in Chapter 2. A working prototype of the n-bit simulator was achieved by way of the FORTRAN level approach. This approach and the resulting system design are discussed in Chapter 3. Chapter 4 briefly summarizes the effects that the Structured Design techniques had upon the system design. The next chapter summarizes the testing performed to test, debug, and validate the n-bit simulation system. Finally, Chapter 6 draws some conclusions and makes recommendations concerning the n-bit simulation tool.

II. COMPASS Analysis And Design

The first approach to solving the n-bit simulation problem was at the COMPASS assembly language level. This approach attempted to accomplish the numerical effects of n-bit wordlength upon an algorithm through modification of COMPASS code. The first section of this chapter explains why the COMPASS level approach was undertaken prior to the FORTRAN level approach. The following sections describe an analysis of the COMPASS code (produced by FORTRAN compilation) with respect to the n-bit simulation problem. Finally, the approach taken to solve the n-bit simulation problem and the major problems encountered which resulted in the failure of the COMPASS level approach are described.

The COMPASS level approach was anticipated to be less complicated than the FORTRAN level approach to the n-bit simulation problem for a number of reasons. In a single FORTRAN language statement several types of arithmetic operations and several data types may be mixed together in many different ways. Each data type would have to be handled differently, and the precedence by which the arithmetic operations are supposed to be performed would require further analysis and handling. Also, arithmetic expressions may occur in almost any type of FORTRAN statement, which could seriously complicate n-bit simulation performance.

On the other hand, at the COMPASS level, each arithmetic operation could be accomplished by a single instruction

involving only equivalent data types. The arithmetic precedence order of execution and conversion of data types has already been accomplished by the FORTRAN compiler. Furthermore, in contrast to many other assembly languages, COMPASS does not include base addressing and relative addressing. This greatly simplifies the insertion of additional COMPASS instructions into the original code, which would enable each arithmetic instruction to be performed while simulating the n-bit wordlength effects.

N-bit Simulation Accomplishment

N-bit simulation for an entire COMPASS coded algorithm can be achieved by simulating the effects of n-bit wordlength for each arithmetic instruction. A typical COMPASS fixed point arithmetic instruction is IX7 X3+X4. This instruction adds the contents of register X3 to the contents of register X4 and stores their sum into register X7. To simulate the effects of n-bit wordlength, the contents of registers X3 and X4 would be modified to contain the equivalent n-bit wordlength values (assuming they have not already). Then the two registers would be added together and the resulting contents of the X7 register would be similarly converted to the n-bit equivalent.

Fixed Point N-bit Simulation

N-bit simulation is accomplished differently for fixed point and floating point number representations. The range of fixed point values which could be represented is limited

by the number of bits in the computer word. An n-bit fixed point word would retain only the right-most (least significant) bits of the CDC 60-bit word. For example, the 16-bit wordlength equivalent of the 60-bit value shown in (a) of Figure 2 is shown by the shaded area in (a) and in (b).

N-bit wordlength representation is realized for fixed point quantities by comparing the 60-bit value to the maximum positive and maximum negative values representable with n-bits, shown respectively in (c) and (d) for 16-bit words. For the purposes of this thesis, whenever a fixed point value exceeds one of these maximums, an overfow condition has occurred and the overflow value is replaced with the maximum value that was exceeded. In Figure 2, (e) contains a negative value with greater than 16 bits of significance. The n-bit simulator would react by replacing it with the value shown in (d).

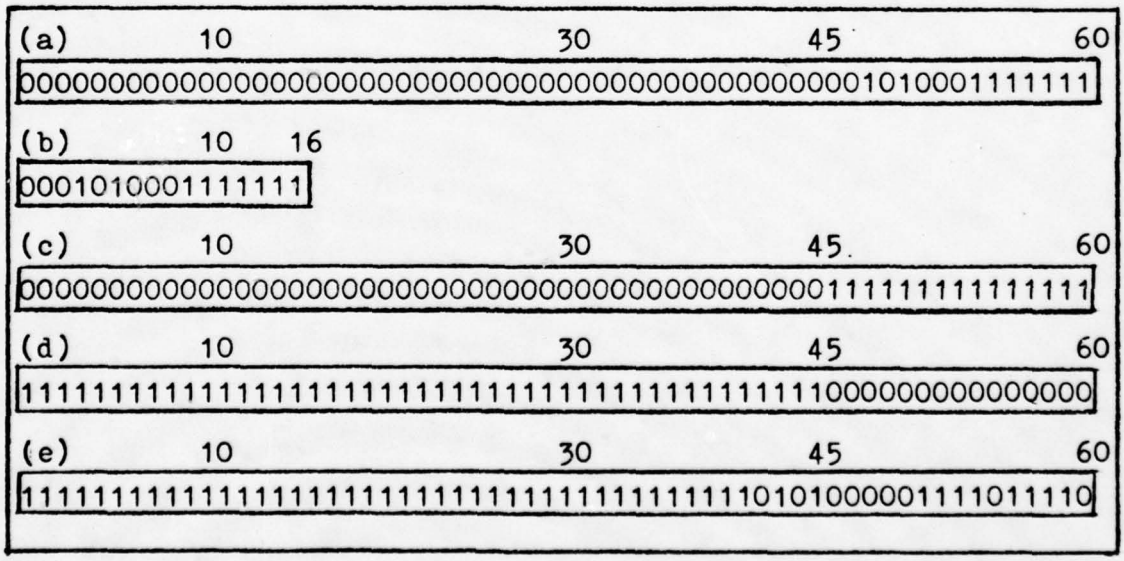


Figure 2 Fixed Point N-bit Simulation

Floating Point N-bit Simulation

To n-bit simulate a floating point word type, the correct number of significant bits for the mantissa must be maintained and the exponent should not be allowed to exceed the equivalent exponent range representable in an n-bit floating point word. CDC floating point data types have normalized mantissa. Therefore, in order to maintain the specified number of significant mantissa bits, only the number of bits specified for it would be retained immediately to the right of the least significant exponent bit (bit #12). This can be accomplished by right shifting (with sign extension) the contents of the word followed by a circular left shift. Right shifting with sign extension displaces the contents of a word X positions to the right, where X is the number of bits computed from subtracting the specified number of mantissa bits from 48. Forty-eight is the length of the CDC 60-bit floating point word mantissa. Those bits shifted off the right-end are lost. All positions vacated on the left by the right shift are replaced with the value of the sign bit. Figure 3 (a) shows an example of a 60-bit floating point word value and (b) shows the resulting contents of (a) after a right-shift of 40-bits (for an 8 bit mantissa). In a circular left-shift, bits shifted off the left end of the word replace those from the right end. In this manner, the significant bits that had been shifted off the right end from the original word are replaced with zeroes for positive numbers and ones for negative numbers. Therefore, for this

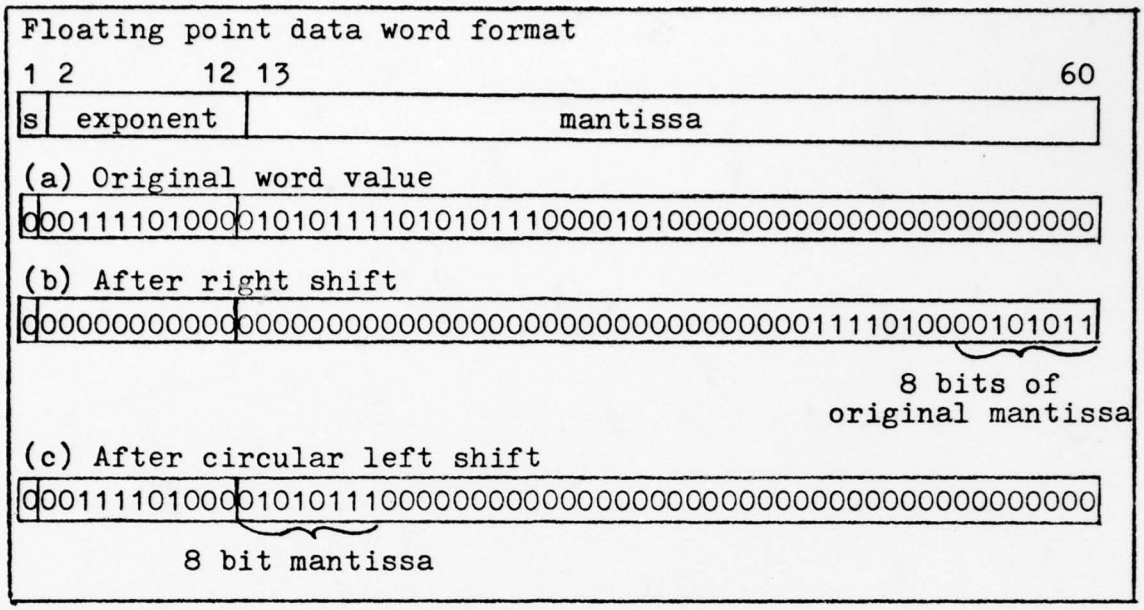


Figure 3 Floating Point N-bit Simulation

example, the only significance for the mantissa is in the left-most 8 bits of the CDC floating point mantissa. Figure 3 (c) shows the resulting contents of (b) after a circular left shift of 40 bits. Using this technique, the value shown in (a) can be modified to contain the same number of significant mantissa bits that an n-bit floating point word with an 8 bit mantissa would contain, this equivalent representation is shown in (c).

Some significance will almost always be lost for decimal number representations, since decimal numbers can not usually be exactly represented with binary storage. For example, the 60-bit octal representation of 3.33 is shown in Figure 4 (a). The equivalent representation for an n-bit floating point word with a 15 bit mantissa is shown in (b). The right most 33 bits of significance have been lost and

(a)	1721652525237366553
(b)	1721652525000000000

Figure 4 N-bit Decimal Representation

the value left is equal to 3.32971; a significance of .00029 has been lost in the 15-bit mantissa representation.

Overflow of the exponent could be detected by comparing the maximum/minimum values possible for a floating point word with the specified bit lengths for the exponent and mantissa. Again overflow detection would result in the maximum/minimum value being stored over the old value. In addition to the case where the exponent gets too large, the case of exponent underflow must also be addressed. This could occur when a value is computed which is so small that its most significant digit is not within the range of the specified exponent (although it is within the CDC's large range). For example, 5 bits may be specified for the n-bit floating point exponent and 6 bits for the mantissa. This exponent could represent an exponent range of ± 16. Therefore, the maximum values representable with that floating point word would be ± 63 (the maximum mantissa value) times 2^{+16} if the implied decimal point is to the right of the least significant mantissa bit.

If the implied decimal point is to the left of the mantissa, the maximum would be .984375 times 2^{+16} (shown in Figure 5). The minimum significance representable would be ± 32 times 2^{-16} and $\pm .5$ times 2^{-16} respectively for the two

different decimal point positions. Underflow for the 12-bit floating point word would occur if a 60-bit division resulted in a quotient less than 2^{-24}. This value would not be within the range representable by a word with a 5 bit exponent. Zero would replace the value that overflowed before the program executed its next operator.

$$01111_{\wedge}111111 = .984375 * 2^{+16}$$
$$011111111111_{\wedge} = 63 * 2^{+16}$$
$$000000_{\wedge}100000 = .5 * 2^{-16}$$
$$000000100000_{\wedge} = 32 * 2^{-16}$$

Figure 5 N-bit Floating Point Value Range for n Equals 12

Now that the strategy has been defined for simulating the effects of n-bit wordlength for arithmetic instructions and the fixed or floating point word values involved, the following section will describe the arithmetic instructions which the n-bit simulation would be performed upon.

The Arithmetic Instructions

Many COMPASS code listings, resulting from the three FORTRAN optimizing compilers, were examined to investigate the handling of FORTRAN assignment statements and arithmetic expressions. The arithmetic operations addition, subtraction, multiplication, and division were found to be handled by the COMPASS instructions in Table I, where Xi, Xj, and Xk may be any combination of the eight X registers (X1 - X8).

Operation type	COMPASS Instruction		Octal code for operation type
	COMPASS mnemonic	Address field	
fixed pt add	IXi	Xj + Xk	36
floating pt add	FXi	Xj + Xk	30
fixed pt subtract	IXi	Xj - Xk	37
floating pt subtract	FXi	Xj - Xk	31
fixed pt multiply	DXi	Xj * Xk	42
floating pt multiply	FXi	Xj * Xk	40
fixed pt divide	*FXi	Xj / Xk	44
floating pt divide	FXi	Xj / Xk	44

*fixed point values in registers Xj and Xk are converted to their floating point equivalents before the division is performed. The quotient Xi is then converted back to a fixed point representation.

Table I Fixed And Floating Point Arithmetic Instructions

Each fixed and floating point single precision arithmetic operation generated by the FORTRAN compiler was performed by a unique instruction with the exception of division. As noted in Table I, fixed point division is performed by a sequence of instructions. The divisor and dividend are converted to their floating point equivalents before a floating point divide is performed. The quotient then is converted back to its fixed point equivalent representation. This sequence of instructions for the fixed point quotient X1=X2/X3 is shown below:

```
PX2  B0,X2  [Pack X2]                    }
PX3  B0,X3  [Pack X3]                    } Converting X2 and X3 to
NX0  B0,X3  [Normalize X3]               }      floating point
FX1  X2/X0  [Divide]
UX1  B7,X1  [Unpack quotient]            }
LX1  B7,X1  [Shift to integer position]  } Converting result back
                                         }      to fixed point
```

Since fixed point and floating point divisions share a common instruction (FXi Xj/Xk) and each type requires a different method to simulate the effects of n-bit wordlength, a choice would have to be made not to allow one or the other of the division types to occur at the FORTRAN program level. Floating point was expected to be the more common usage for algorithms, so fixed point divisions would be restricted from n-bit simulated algorithms. If the fixed point division sequence always appeared as consecutive instructions, fixed and floating point division could be differentiated; but the FORTRAN optimizing compilers order the COMPASS instructions for the most efficient execution, and consequently the COMPASS code generated for a sequence of FORTRAN statements may be completely intermixed.

One other potential problem could occur for the fixed point multiplication instruction DXi Xj*Xk shown in Table I. DXi Xj*Xk is actually the COMPASS double precision multiply instruction. However, this conflict of usage would not occur since the algorithms for the n-bit simulator would not require double precision FORTRAN variables (Assumption 5 states this in Chapter I).

Now that the COMPASS instructions, which perform the arithmetic computations have been described, two methods of incorporating modifications for each arithmetic instruction will be described in the following section.

Two Approaches to Incorporating N-bit Simulation

Two fundamental approaches to implementation of the n-bit simulation modifications were considered. The first approach would replace each arithmetic instruction with subroutine calls. The second approach would substitute, in-line, the necessary instructions to perform the n-bit simulation effects.

The subroutine approach would require an argument list to accompany each subroutine call indicating the registers involved with that instruction. This argument list would key the handling subroutine to manipulate the designated registers. Uniquely identifying each instruction/register combination with a separate subroutine would require 512 different subroutines per arithmetic instruction type (three registers are used for each arithmetic instruction and each register may be any one of eight possible X registers, making 8^3 or 512 possible combinations). Clearly that would have been too many routines. So an argument list would be necessary indicating the registers involved, and a technique would have to be devised in each subroutine to handle them correctly.

In contrast, the in-line code approach would be simpler and save subroutine linkage time, but would require more core space. If branching would be necessary within the substituted code sequence, some method would have to be devised to provide unique labels within each substituted sequence. For the initial COMPASS level investigation, the in-line approach was chosen.

Methods for Isolating Arithmetic Instructions

In both cases previously stated, the first obstacle was trying to isolate arithmetic operation instructions of Table I from the rest of the COMPASS code. Three techniques were considered.

The first technique involved examining the raw COMPASS object code (machine language). This would be a complicated procedure due to problems involved distinguishing data from instructions. Also labels, already assigned to fixed locations, are transparent from their associated object code. Injecting additional COMPASS object code to perform n-bit simulation would lead to misalignment of labels with respect to their associated object code locations. When examining the object code, a table containing all possible COMPASS instruction codes and the associated instruction length would have to be maintained. In addition, one must account for no-operation instructions (NOOPS). NOOPS are used to fill in remaining portions of 60-bit words which are not completely filled by instructions due to labels or other conditions. It would be a cumbersome task indeed. Consequently, this technique was quickly ruled out.

The second technique involved examining the COMPASS source code generated from the FORTRAN compilation of the algorithm. Distinguishing the arithmetic operation instructions would require a table lookup of each COMPASS code mnemonic. When matches occurred, a sequence of instructions would be inserted to perform the n-bit wordlength effects

for that instruction type. A significant advantage of this method is that the COMPASS assembler would do the work of realigning all COMPASS object code and label pointers. COMPASS source code would be accessed by using the E option on the FTN control card. The sequence of control cards in Figure 6 could be used to access the COMPASS source code generated from a FORTRAN compilation. The execution of NBIT (in Figure 6) would modify the original COMPASS source code (the COMP file) for n-bit simulation producing the revised n-bit simulated code in file COMP2. COMP2 would then be assembled and executed.

```
FTN, E = COMP.
NBIT, COMP,,, COMP2.
REWIND, COMP2.
COMPASS, I = COMP2, S = FTNMAC, B = BIN.
LDSET (LIB = FORTRAN/SYSIO).
BIN.
```

Figure 6 N-bit Simulation Control Card Sequence

The third method for isolating and handling the arithmetic instructions would take advantage of some sophisticated features of the COMPASS assembler called operation definitions (OPDEFs) and IF conditionals (IFCs). Both are pseudo instruction types. The OPDEF consists of a sequence of COMPASS source code that is saved and assembled when an instruction type within a given routine matches the syntax specification. It usually includes parameters to be substituted for formal parameters, so that the object code generated can vary with

each assembly of the definition and whatever combination of registers that are involved in the instruction (REF 3: 5-27). Using this feature, the assembler would do the work by replacing the arithmetic instructions with the n-bit simulation sequence.

Figure 7 shows a COMPASS routine in which an OPDEF was inserted, then assembled with the rest of the COMPASS code. This particular OPDEF syntactically defines the general instruction type IXi Xj+Xk. P1, P2, and P3 used within the OPDEF would represent the registers Xi, Xj, and Xk respectively. In the sequence of original COMPASS code, the IX7 XO+X4 instruction would match this syntax and is replaced by this sequence when it is assembled as shown in the after assembly code sequence. After assembly, the object code (shown on the left) is produced by the COMPASS assembler upon detection of the IX7 XO+X4 instruction. It can be seen after the assembly that the OPDEF for the fixed point add instruction replaced the original one.

Two additional pseudo instructions were also used in this example: SET (REF 3: 4-36) and CPOP (REF 3: 6-7). CPOP was used to disguise the fixed point addition instruction from its own OPDEF. Otherwise, if an instruction were used within its own OPDEF definition it would result in the assembler getting caught in an infinite loop of replacing an instruction with itself, which triggers the process again, and so on.

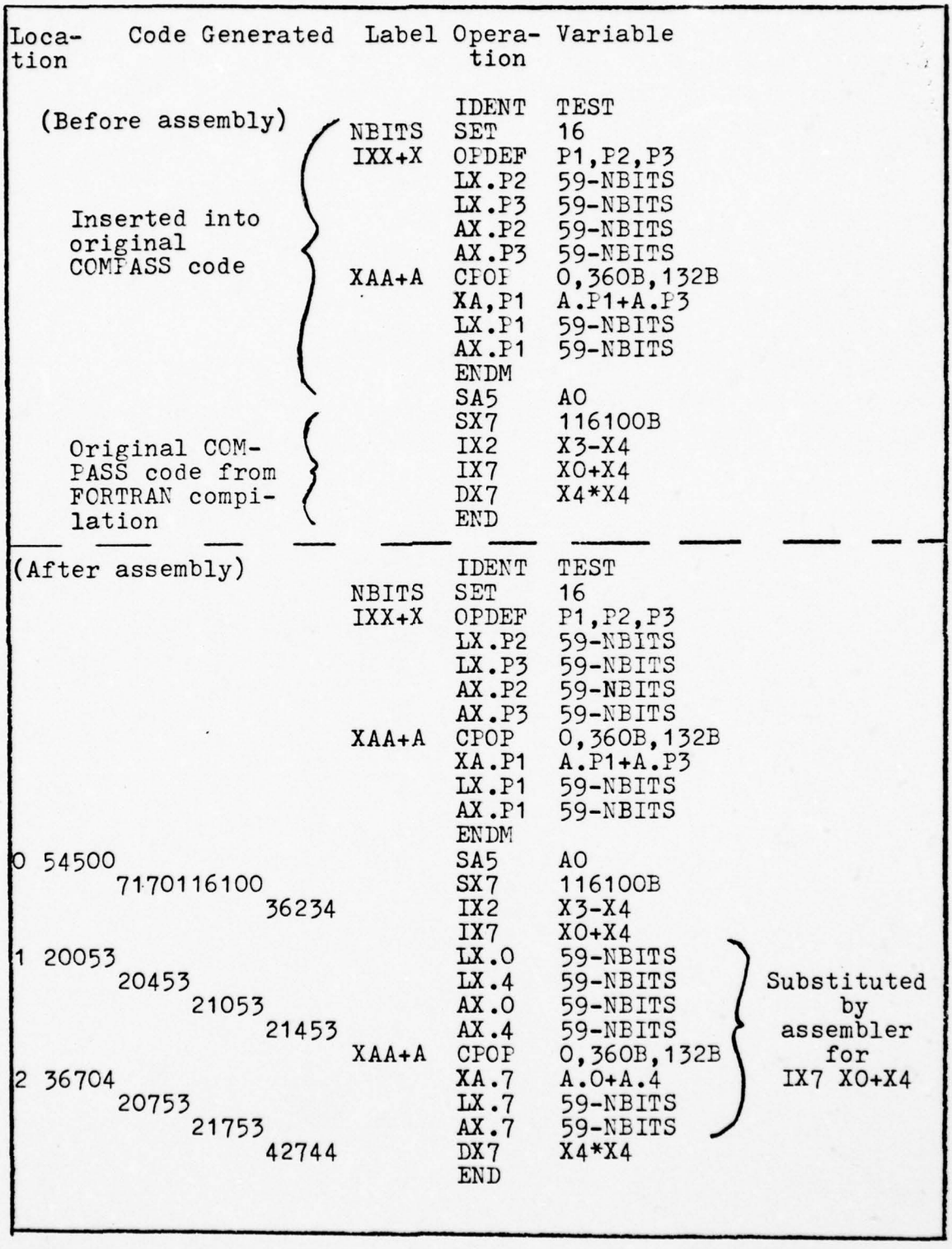


Figure 7 OPDEF Application

The other pseudo instruction mentioned earlier, IFC, could be used to incorporate the user options, for example, truncation or rounding effects. Each effect would require somewhat different code. Both code sequences could be incorporated into the arithmetic instruction OPDEF, but only the one whose option was on would be invoked as part of the OPDEF, upon assembly.

This package of COMPASS pseudo instructions could greatly simplify the n-bit simulation set up. However, a means would have to be devised to get the OPDEFs inserted within each routine, since they are effective only for the routine in which they were defined. If branching were required within an OPDEF to perform n-bit simulation effects, a special element called *L (REF 3: 2-9) and the SET pseudo could be used within each OPDEF to offer each invocation of the OPDEF a unique label. Figure 8 shows an example of its usage within an OPDEF. The SET *L + 2 instruction would put the correct location into the symbol table for N (shown at location 15). When ZR X.P1,N referenced it, the value (a location) currently in the symbol table would be put in the object code. The SA.P1 P,1 instruction (where the label is actually needed) would be positioned a known number of locations down from the

```
Location    Label     Instruction
   15         N       SET *L + 2
                      ZR X.P1,N  (wants to jump to
                                          SA.P1 P,1)
   16                 (other instructions)
   17                 SA.P1 P,1    [needs the label]
```

Figure 8 Example Of Labeling Within An OPDEF

SET instruction. Since each OPDEF invocation would occur at a different location, each reference would be unique, since the value of N would change with the location of each invocation.

Problems Encountered With The COMPASS Level Approach

After further analysis of the COMPASS level approach, three problems were uncovered. These problems will be discussed in the following subsections.

Multiples Of Two

Upon further analysis, it was discovered that when the FORTRAN optimizing compiler 1 or 2 was used, fixed point multiplications and divisions by multiples of two were not converted into arithmetic multiply or divide instructions, but were accomplished by shifting the register contents left or right the number of bits required to obtain the proper result. However, the zero level compiler did convert such occurrences into the expected arithmetic instructions. Therefore, a user would either have to compile the algorithm using the level 0 optimizer which would result in less efficient execution or be restricted from using multiples of two in arithmetic equations, since those arithmetic operations could not be n-bit simulated. The user could still use a multiple of two in an equation for optimization level 1, but not as a constant within the equation. It could overcome the optimizers intelligence by assigning that multiple of two (constant) to a variable just prior to the expression as shown

below. This procedure would force the compiler to generate the COMPASS arithmetic instruction.

(desired equation: IRSLT = I1/8 * I2)

```
ITEMP = 8
IRSLT = I1 / ITEMP * I2
```

The problem with multiples of two could be overcome by putting an added restriction upon the user. However, the problems described in the following subsections are of a more serious nature and could not be resolved.

No Double Precision Arithmetic Accuracy

Another problem that became evident quickly was that no FORTRAN arithmetic expression could be evaluated in double or triple precision, then stored as a single precision value. This is because at the COMPASS level the last arithmetic operation of a compiled FORTRAN equation is indistinguishable from the first. There is no way at the COMPASS level to detect the first operation generated from a FORTRAN arithmetic statement. The COMPASS code generated from individual FORTRAN statements is transparent at the COMPASS level and in many cases the code from several FORTRAN statements are intermixed together so the computer can execute with greater efficiency. Thus all operations would be limited to single precision and that user option specified in the requirements would not be available.

Requirements For Spare Registers

A much more serious problem evolved when it was determined

that extra registers would be needed, in addition to the ones involved in the arithmetic instruction, in order to perform the required n-bit simulation effects. These extra registers would be required for storing the maximum and minimum values of the n-bit fixed and floating point words. To determine overflow conditions these values would have to be compared to the contents of the registers involved in the arithmetic instruction. In order to make the comparisons, the values must be stored in a 60-bit register. However, due to the nature of the COMPASS level approach, nothing can be presumed to be known about any register before that instruction or after. At the time the arithmetic instruction is executed, any or all of the 16 X and B registers may contain values which were set up by the FORTRAN compiler to be used later. Modification of a register's contents could be disasterous.

Since the status of all registers outside the ones involved in the arithmetic instruction is not known, all registers not involved in the arithmetic operation instruction must have their values restored to what they were just prior to the operation. Either a register saving/restoring technique would have to be employed or else some register with known or fixed values would have to be found which could be used as an extra register then restored afterwards.

Several possibilities were investigated and some techniques were devised, but none which were reliable 100% of the time. The different avenues taken are described more fully in Appendix A. Literature searches in CDC manuals

could not produce any amount of detail about the register handling conventions of the FORTRAN compilers. Nor could any register saving/restoring convention be found. Apparently, the compilers do not save all registers before jumping to external subroutines, as many computer systems do (e.g. IBM). Instead it saves only those registers it knows through its intelligence that it will need later.

Without extra registers, no rounding effects could be performed and no overflows could be detected. It logically follows that without overflow detection, no overflow messages could be printed nor would there be a way or means to substitute maximum values for values that overflow. The only action that could be performed would be to limit fixed point values and floating point mantissas to the specified number of bits of precision (through left and right register shifts, to accomplish the truncation).

Conclusions

The more the COMPASS level approach was explored, the more limitations and shortcomings were uncovered. Single assignment statements could not be detected, double and triple precision operations could not be performed as options, and no completely reliable scheme could be devised to generate extra registers required for overflow detection and handling. Therefore, the COMPASS level approach was aborted. The n-bit simulation problem was next addressed at the FORTRAN level as discussed in the following chapter.

III. FORTRAN Analysis And Design

After failing to establish an operational n-bit simulation at the COMPASS level, a FORTRAN level approach to the n-bit simulation problem was endeavored. The FORTRAN level approach attempted to accomplish n-bit simulation by modifying the user's FORTRAN-programed algorithms. This chapter will describe an analysis of the FORTRAN language with respect to the n-bit simulation problem. Also it will describe the features of the FORTRAN algorithm preprocessor and n-bit simulation subroutines that were designed to accomplish the n-bit simulation.

FORTRAN Problem Analysis

A basic assumption for this thesis was that n-bit wordlength effects could essentially be simulated by modifying only the arithmetic expressions within a program. This is because the n-bit simulation tool was devised to enable a user to evaluate the numerical stability and precision of algorithms as a function of wordlength. In Standard ANSI FORTRAN, arithmetic expressions may appear in almost all statement types. N-bit simulation of arithmetic expressions for all possible statement types would complicate the FORTRAN level approach to a considerable extent. Therefore, it was decided that the initial FORTRAN n-bit simulation tool developed would limit n-bit wordlength effects to only those expressions occurring with FORTRAN assignment statements.

This would be the minimum requirement to n-bit simulate an entire algorithm and would still not unduly inhibit the user from programing in his normal manner.

To simulate the effects of n-bit wordlength for an arithmetic statement, each arithmetic operation and its associated operands would, first, have to be isolated from the rest of the statement. Then each operand would have to be modified so that it represents the n-bit wordlength equivalent (if it has not already). Next, after performing the operation, the result must be similarly modified. If the operation would have resulted in an overflow condition for the n-bit computer, an overflow message should be printed to the user and the overflow value should be replaced with the maximum value representable with an n-bit word (as specified in the n-bit simulation requirements of Chapter I).

To correctly evaluate an arithmetic statement, the order of precedence in which the FORTRAN arithmetic operations are normally performed would have to be preserved. Also the variable types (fixed and floating point) would have to be determined then handled appropriately and labels accompanying arithmetic statements would have to be handled so that the execution sequences of the algorithm would not be affected.

Figure 9 shows the Backus Normal Form (BNF) for all possible syntactic representations of an arithmetic assignment statement. Any arithmetic assignment statement may consist of unaries, integer (fixed point) and real (floating

```
<arithmetic statement> ::= <identifier> = <expression>

<identifier>           ::= <letter> | <identifier> <letter>
                           <identifier> <digit> | <function>

<expression>           ::= <sign> <term> | <expression> <sign'>
                           <term>

<term>                 ::= <factor> | <term> <mul-op> <factor>

<factor>               ::= <f2> | <factor> <exponentiation>
                           <f2>

<f2>                   ::= <identifier> | <literal> |
                            ( <expression> )

<function>             ::= <identifier>  ( <arguments> )

<arguments>            ::= <expression> | <arguments> ,
                           <expression>

<literal>              ::= <I2> | <real #>

<real #>               ::= <I2> . | <I2> . <I2> | . <I2>

<I2>                   ::= <digit> | <I2> <digit>

<letter>               ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|
                           Q|R|S|T|U|V|W|X|Y|Z

<digit>                ::= 0|1|2|3|4|5|6|7|8|9

<sign>                 ::= <sign'> | <null>

<sign'>                ::= +|-

<mul-op>               ::= *|/

<exponentiation>       ::= **
```

Figure 9 Backus Normal Form For Arithmetic Statement

point) variables, integer and real literals, arithmetic operations, parentheses, array items, and function subroutine calls. Some examples of various arithmetic assignment statements are shown in Figure 10. Boolean IF statements may also contain arithmetic assignment statements. The arithmetic statement would be executed if the logical or relational expression in the IF statement predicate is true. The basic form of a Boolean IF statement with an arithmetic statement is:

IF (Logical or relational expression) <arithmetic stmt>

```
Label  Col.
field  7

       A = B*C*(LONG1-5.324/(-24))
 1     B = FUNC(4*J,-K,F2-LAST)+NUMB1
       C = -W**5
 25    D = ARRAY1(5,I)*5.014327
       E = +(0.5**3**TWO)/DIVDER
 4     IF(A.EQ.(10*R))F = 2000+45.+.3*REALNUM
       RESULT = FINAL
```

Figure 10 Examples Of Legal Arithmetic Statements

Evaluation Of An Arithmetic Statement Through Reverse Polish Notation

The task of simulating the effects of an n-bit wordlength machine requires that the result of a computation for an arithmetic statement be modified to reflect its n-bit wordlength equivalent. This necessitates the isolation of each arithmetic operation and the operands the operation is acting upon.

Therefore, each FORTRAN arithmetic statement must be broken down into a series of arithmetic statements, each performing a single arithmetic operation. The final result from the series of single computation statements would be algebraically equal to the result computed from the whole, complex arithmetic statement from which they were derived.

An example of breaking a complex statement into a series of single computation statements is shown in Figure 11. Equation (a) of Figure 11 could be accomplished just as well through the sequence of single operation arithmetic statements shown in (b). However, the intermediate variables used in (b) must agree with the dominant data type (mode) of each computation. That is, when an expression contains both fixed and floating point quantities, the final result would have to be of the floating point data type since floating point is dominant over fixed point in FORTRAN.

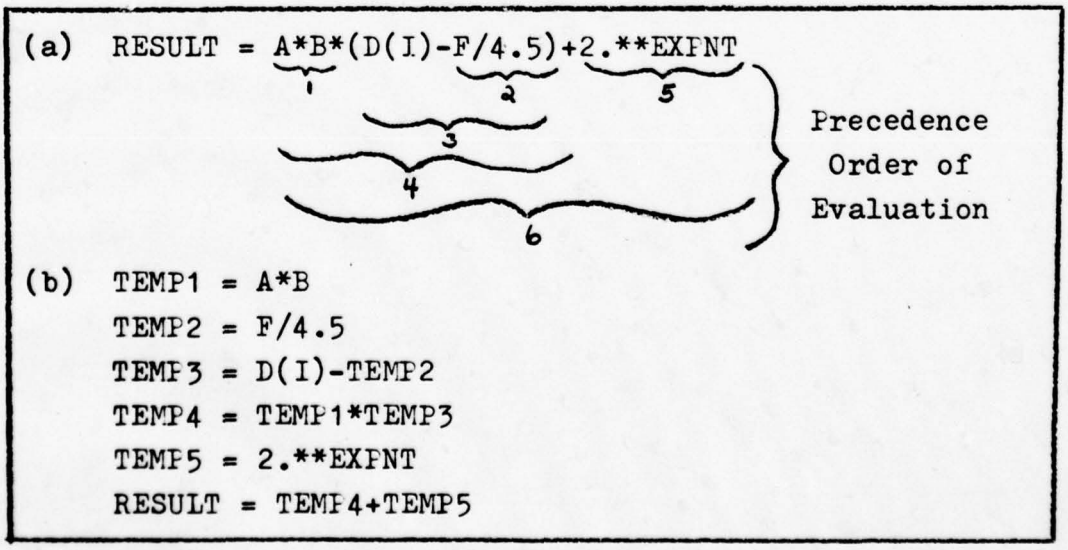


Figure 11 Equivalent Computation Of A Complex Arithmetic Equation

When a FORTRAN compiler analyzes FORTRAN arithmetic statements, it determines the proper sequence in which the computations should be performed and the necessary mode conversions needed for the differing variable data types. Then the compiler generates the assembly language instructions which will do the job; forming intermediate results between computations until the final assignment is made.

An approach similar to one a FORTRAN compiler might use for evaluating an arithmetic statement is the Early Operator Reverse Polish Notation (REF 5: 55). Polish notation is a mathematical notation that provides for the representation of complex expressions in a non-ambiguous manner, without relying on hierarchial delimiters such as parenthesis. The basic difference between standard mathematical notation and Polish notation is the relative position of the operator to its operands.

Figure 12 shows an example of a FORTRAN arithmetic statement and its equivalent Reverse Polish Notation representation. To evaluate the Polish string, it is scanned left to right until an operator is found. Then that operator is used on the preceding two operands-replacing the two

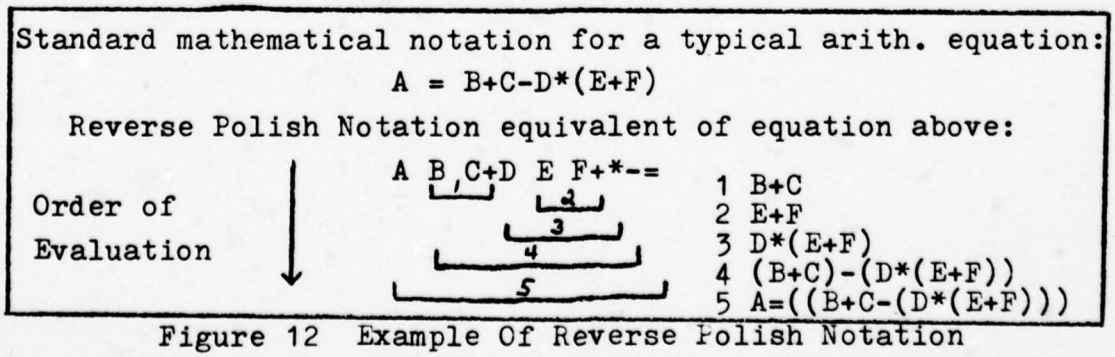


Figure 12 Example Of Reverse Polish Notation

by the result and continuing the scan. Each arithmetic operator has a hierarchial precedence value. Table II shows the precedence relationship among the arithmetic operations of the FORTRAN language. Operations with higher precedence in the hierarchy are performed first in the evaluation of an arithmetic equation. Operations of equal precedence are evaluated left to right. The BNF for FORTRAN arithmetic statements in Figure 9 also reflects this precedence of the operators.

	Arithmetic operation	Precedence rank
Higher precedence ↓	+	1
	-	1
	*	2
	/	2
	- (unary)	3
	+ (unary)	3
	**	4

Table II Operator Precedence Order For Evaluating Arithmetic Equations

Through the use of Polish Notation techniques, the process of generating a sequence of single operator statements from a single complex arithmetic statement is greatly simplified. Once an arithmetic statement is in its equivalent Polish Notation form, the Polish string can be scanned left to right for operators. For each operator a function subroutine call is generated with the operands involved passed along as arguments. The subroutine called would perform the computation while simulating the effects of n-bit wordlength.

Additional FORTRAN Language Considerations/Problems

In the analysis of the FORTRAN language with respect to the n-bit simulation problem, several obstacles and potential problems had to be dealt with to handle all cases of arithmetic statements. Besides the variables and arithmetic operations, an arithmetic statement may involve function subprogram references, arrays, labels, and IF statements. An additional consideration is to provide the user a means of tracing down overflow occurrences. These cases will be discussed in the following subsections.

Function References. Function subprogram references with a FORTRAN arithmetic statement present a potential problem since the functions they reference may not be coded in FORTRAN. To be n-bit simulated completely, all code executed must be written in the FORTRAN language. System functions such as SIN, COS, and ATAN (external or FORTRAN library functions (REF 4, I-8)) are not coded in the FORTRAN source language and therefore can not be altered by the FORTRAN level approach to simulate n-bit wordlength effects. A conversion of all intrinsic and external functions into their FORTRAN language equivalents would remedy this problem. However, without a FORTRAN version, the most effective action that can be done to approximate n-bit wordlength affects on an intrinsic function would be to n-bit simulate each argument of the function and then n-bit simulate the function value returned.

Arrays. N-bit simulation for each argument (subscript) of a FORTRAN array item would not really serve any purpose. FORTRAN array arguments must always be fixed point values. They are usually relatively small fixed point values and should be well within the range of an n-bit wordlength computer's fixed point words, in presumably every case. Consequently, it was considered unnecessary overhead to n-bit simulate expressions and single arguments used for array subscripts. Treating the entire array item the same as a single variable was found sufficient.

Since arrays and function subprogram references may have the same syntactical representation in an arithmetic statement but would be n-bit simulated differently, they must be differentiated from each other. To distinguish arrays from functions, each program unit is examined to determine the variables declared as arrays. Arrays may be declared in either DIMENSION, INTEGER, REAL, or COMMON declaration statements. So for each variable in question, all array names declared within that program unit are compared to that variable name. If the variable in question is not an array, then the variable defaults to being treated as a function subprogram reference, and n-bit simulated accordingly.

Labels. Another potential problem area for FORTRAN is where a label is attached to an arithmetic statement which is the last statement to be executed in the DO loop. A label in FORTRAN can be used for two different purposes.

These usages of a label will be described before defining the problem in detail.

A statement label uniquely identifies a statement so it can be referenced by another statement. Labels occur in columns 1-5 of a FORTRAN statement. The actual statement appears in columns 7-72.

Statement number	Label	Statement
1		C = X+2.5
* 2		GO TO 38
3	14	A = B
4	38	C = B*D/5

* Execution of statements 1 and 2 would be followed by a branch statement 4 (through the use of label 38)

Figure 13 Normal Use Of A Statement Label

Generally, a label marks a statement so that the statement can be branced to for execution, as shown in Figure 13. However, there is an exception in the case of DO loops. A DO loop statement, such as the one shown in Figure 14, is used to establish a sequence of statements to be executed repeatedly for a specified number of times. In the case of Figure 14, a label is associated with the last statement of a DO loop sequence. This label is called a DO loop termination statement label. Upon execution of its associated statement, program execution returns to the first statement of the DO loop sequence to execute the DO loop sequence again. This looping would continue until something causes the program to branch out of the loop. The main difference

between the DO loop termination label and a normal label is that a termination label is used to indicate the last statement of a sequence to be executed and a normal label is used to indicate the first of a sequence.

Statement number	Label	Statement
1		DO 70 L = 1,20
2		A(L) = A(L)+5
* 3	70	C(L) = D(L)+A(L)

* Label 70 indicates last statement executed in the DO loop. After execution of statement 3, L would be incremented and the loop executed once again, until L exceeds the value 20.

Figure 14 Example Of DO Loop Termination Label

The problem with respect to this effort arises when a complex arithmetic statement is broken up into a sequence of single operation arithmetic statements; especially if the FORTRAN program is analyzed and translated on a statement by statement basis (i.e. each statement is analyzed without knowledge of the context of its use with regard to other statements of the program). Each label type must be handled differently. Figure 15 shows an example of what would happen if a termination label were treated the same as s normal label in the translation of a complex arithmetic statement to its single operation series equivalent. In the example, it is associated with an arithmetic statement. Normal label handling would distort the real meaning of the DO loop. In the translated code of Figure 15, part of the computation is

now positioned outside the DO loop sequence. Thus, the results would not be the same.

Before translation	After translation
DO 5 I = 1,7	DO 5 I = 1,7
5 SUM = A(I)+SUM*2	5 T1 = SUM*2
	SUM = A(I)+T1 ← outside of loop now

Figure 15 Incorrect Translation Of A DO Loop

To assure a consistent meaning of labels for n-bit simulation modifications, the user will always be required to assign DO loop termination labels to CONTINUE statements. A CONTINUE statement can always be used as the last statement in a DO loop sequence without changing the loop's meaning. A CONTINUE statement will never be modified for n-bit simulation since only arithmetic statements are changed. Figure 16 shows how a DO loop termination label could be moved to a CONTINUE statement without changing the performance of the loop. Figure 16 also shows the single operation translation of that loop.

Before translation	After translation
DO 5 I = 1,7	DO 5 I = 1,7
SUM = A(I)+SUM*2	T1 = SUM*2
5 CONTINUE	SUM = A(I)+T1
	5 CONTINUE

Figure 16 Use Of CONTINUE Statement In A DO Loop

IF Statements. Another slight complication of the FORTRAN approach is translating an arithmetic statement contained as part of an IF statement. Figure 17 (a) shows such an IF statement. A procedure had to be developed so that arithmetic statements contained as part of IF statements could be translated into a series of single operation statements without distorting the original program's meaning. Figure 17 (b) restates (a) in such a manner that the arithmetic statement is now separated from the IF statement and could be translated into a sequence of single operation statements, shown in (c), without affecting the original meaning. Two new labels and two GO TO statements were added to the original sequence in (a) to preserve the correct order of execution. The execution paths for (a), (b), and (c) are identical as shown in (d).

Overflow Tracing. Another FORTRAN consideration was assisting the user in tracing down the instances where an overflow of the n-bit wordlength occurred during the n-bit simulation of an algorithm. It would be helpful to the user to indicate the name of the routine and the line number within the routine in which the overflow condition was detected. This would be in terms of the original FORTRAN coded algorithm and not the n-bit simulation modified code. In addition, specifying the operation executed at the time of the overflow might also be helpful.

To assemble the necessary data for these overflow messages, the beginning of each new routine must be detected.

(a)

```
        A = FIRST
        IF(A.EQ.C) RESULT = A/5*B
        A = NEXT
```

(b)

```
        A = FIRST
        IF(A.EQ.C) GO TO 1500
        GO TO 1501
1500    RESULT = A/5*B
1501    A = NEXT
```

(c)

```
        A = FIRST
        IF(A.EQ.C) GO TO 1500
        GO TO 1501
1500    TEMP1 = A/5
        RESULT = TEMP1*B
1501    A = NEXT
```

(d)

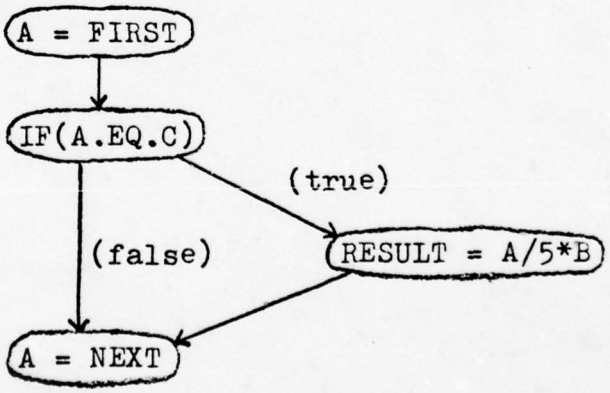


Figure 17 Arithmetic IF Statement Handling

New program units (routines) are initiated by one of the following FORTRAN keywords: PROGRAM, SUBROUTINE, FUNCTION, REAL FUNCTION, or INTEGER FUNCTION. All other routine initiation keywords, such as LOGICAL FUNCTION or DOUBLE PRECISION are not allowed in n-bit simulation programs, by one of the thesis assumptions.

Once the beginning of a new routine is detected the routine name is extracted, the line counter for that routine is initialized to one and incremented with each succeeding statement. When an arithmetic statement is translated into a series of subroutine calls, the routine name and line number containing the arithmetic statement are included as arguments. Therefore, when an n-bit wordlength overflow is detected, this information along with the operation being performed at the time would be printed. Then, the user could easily trace down the statement which was the source of the overflow condition.

The analysis of the FORTRAN language with respect to n-bit simulation now has been described. The following section describes the system developed which incorporated the results of the FORTRAN analysis and enables a FORTRAN algorithm to simulate n-bit wordlength effects.

N-bit Simulation Tool Design

The design of the n-bit simulation tool was divided into to two parts. Part one preprocesses the FORTRAN algorithms before they are executed; translating each FORTRAN arithmetic

statement into a series of subroutine calls, each of which performs one of the statements's computations. Part two of the design develops the subroutines which would perform the n-bit wordlength effects.

Preprocessor

The purpose of the preprocessor is to perform a lexical scan and analysis of each statement in the FORTRAN algorithm, then convert the arithmetic statements into a series of n-bit simulation function subroutine calls.

The preprocessor essentially performs two major functions:

1) It performs a syntactic analysis for arithmetic statements and converts them into their Reverse Polish Notation equivalent representation. (Further detail concerning the syntax analysis is contained in Appendix D.).
2) The preprocessor disassembles the Reverse Polish Notation representation into function subroutine calls, tailored according to the variable data types, and arithmetic operations involved. Each function reference replaces an arithmetic operation from the original statement. If temporary variables are required to store intermediate results, a variable matching the dominant data type involved is created. In addition, for each function reference, the preprocessor supplies the routine name and line number of the FORTRAN statement from which the statement was derived.

The syntax description for an arithmetic statement was shown by the BNF of Figure 9. Each statement fitting this syntax description would be transformed into one or more function subroutine references. Each function reference would replace a single arithmetic operation. Any statement which did not fit the syntax description would not be affected. Figure 18 contains an example of a preprocessed program.

In Figure 18, the temporary variables were created to hold the intermediate computed values. Floating point variables are stored in variables preceeded by RTTTT and fixed point variables by ITTTT. The last letter is incremented from A to B to C and so on for each succeeding intermediate variable within each equation. These names will be restricted from the user, but are named so that they are not likely to conflict with any variable name a user might create.

The function reference names indicate the type of operation that is being performed from the original arithmetic statement. For example, statement 8 of the preprocessed program performs the multiplication of 15 and B from the original program statement 8. The last three letters of the function name indicate the type of operation (e.g. ADD for addition, and MPY for multiply). The first two letters indicate the data type of the operands being passed as arguments. II indicates both operands are fixed point variable data types. RR indicates both operands are floating point. R1 and R2 indicate one of the operands being passed is fixed point and the other floating point. The number 1

```
Statement #     Original FORTRAN Algorithm

    1           PROGRAM SAMPLE (INPUT, OUTPUT)
    2           DIMENSION A(10), K(30), KEY(7)
    3           CALL SETNBIT (24,1,1,1,17,6,0,KEY)
    4           B = 25.3
    5           READ *, L1
    6           L1 = L1
    7           CALL SUB1 (A,25,B)
    8           NEXT = A(1)+15*B
    9           K(L1) = NEXT/50*(A(2)+B)
    10          STOP "END OF SAMPLE"
    11          END

Original        Preprocessed Algorithm Version
Statement #

    1           PROGRAN SAMPLE (INPUT,OUTPUT)
    2           DIMENSION A(10), K(30), KEY(7)
    3           CALL SETNBIT (24,1,1,1,17,6,0,KEY)
    4           B = RASGN(25.3, 7HSAMPLE ,6,1,KEY)
    5           READ *, L1
    6           L1 = IASGN(L1, 7HSAMPLE ,6,1,KEY)
    7           CALL SUB1(A,25,B)
    8           RTTTTAA = R1MPY(15,B, 7HSAMPLE ,8,0,KEY)
                NEXT = RRADD(A(1),RTTTTAA,7HSAMPLE ,8,1,KEY)
    9           ITTTTAA = IIDVD(NEXT,50,7HSAMPLE ,9,0,KEY)
                RTTTTAA = RRADD(A(2),B,7HSAMPLE ,9,0,KEY)
                K(L1) = R1MPY(ITTTTAA,RTTTTAA,7HSAMPLE ,9,1,KEY)
    10          STOP "END OF SAMPLE"
    11          END
```

Figure 18 Example Of A N-bit Simulated Algorithm

in R1 indicates the fixed point variable is the first argument of the function, 2 indicates it is the second. Table III contains a list of all function subroutines. It also indicates the arithmetic operation performed and the operand data types involved.

Name of Function	Operation	Operand 1 Data Type	Operand 2 Data Type
IIADD	+	I	I
R1ADD	+	I	R
R2ADD	+	R	I
RRADD	+	R	R
IIMNS	-	I	I
R1MNS	-	I	R
R2MNS	-	R	I
RRMNS	-	R	R
IIMPY	*	I	I
R1MPY	*	I	R
R2MPY	*	R	I
RRMPY	*	R	R
IIDVD	/	I	I
R1DVD	/	I	R
R2DVD	/	R	I
RRDVD	/	R	R
IIEXP	**	I	I
R1EXP	**	I	R
R2EXP	**	R	I
RREXP	**	R	R
IASGN	=	I	-
RASGN	=	R	-

I - Integer (Fixed Point)
R - Real (Floating Point)

Table III N-bit Simulation Function Subroutine Names

There are two forms of n-bit simulation functions: single operation replacements and simple assignment statement replacements (e.g. IASGN and RASGN). The general format of these two function types are:

```
FUNCTION (OPERAND1,OPERAND2,ROUTINE,STMT#,FINALFLG,KEY)
FUNCTION (OPERAND,ROUTINE,STMT#,FINALFLG,KEY)
```

The OPERANDS are variables which will be n-bit simulated by the function. The ROUTINE is the name of the routine containing the statement and STMT# is the line number of the routine in which the statement occurs.

FINALFLG is used to distinguish intermediate value assignments from final assignments of an arithmetic assignment statement. Zero indicates the result is an intermediate assignment, one indicates a final assignment. Intermediate assignment results will maintain the arithmetic precision specified by the user; either single, double, or triple precision. Final assignments indicate the final computation of the arithmetic statement. The resulting value from a final computation always has single precision accuracy.

The last argument of the function is KEY. KEY is a seven item array containing values computed by the SETNBIT subroutine and is used by the n-bit simulation subroutines to simulate n-bit wordlength effects.

Each arithmetic statement in its original form appears in a COMMENT statement (a non-executable statement type). This COMMENT is printed just prior to the series of function

calls derived from it (Figure 19). This procedure helps improve readability of the preprocessed program code for the user.

```
Stmt. #    Original Algorithm Statements:
  1        A = B+FUNC(A,5*K,K1)   where FUNC is a FUNCTION
  2    51  IF(A.EQ.C) LAST = FINAL/R*S*25.97
-------------------------------------------------------------
           Preprocessed Algorithm Statements:
       C   A = B+FUNC(A,5*K,K1)                          <--
           RTTTTAA = RASGN(A,7HPROGRM1,1,1,KEY)
           ITTTTAA = IIMPY(5,K,7HPROGRM1,1,1,KEY)
  1        ITTTTAB = IASGN(K1,7HPROGRM1,1,1,KEY)
           A = RRADD(B,FUNC(RTTTTAA,ITTTTAA,ITTTTAB),
          *        7HPROGRM1,1,1,KEY)
       C   IF(A.EQ.C) LAST = FINAL/R*S*25.97            <--
       51  CONTINUE
           IF(A.EQ.C) GO TO 7901
  2        GO TO 7902
    7901   RTTTTAA = RRDVD(FINAL,R,7HPROGRM1,2,0,KEY)
           RTTTTAB = RRMPY(RTTTTAA,S,7HPROGRM1,2,0,KEY)
           LAST = RRMPY(RTTTTAB,25.97,7HPROGRM1,2,1,KEY)
    7902   CONTINUE
```

Figure 19 Preprocessor Output For Two Complex Arithmetic Statements

A brief description of the preprocessor's operation has now been given. A detailed description of the preprocessor is given in Appendix C. Chapter 5 contains a description of the preprocessor design and summarizes the functions of the modules within the preprocessor. The following section describes the n-bit simulation subroutines which are called by the n-bit simulated algorithm built by the preprocessor.

N-bit Simulation Subroutines

The n-bit simulation function subroutines are designed to perform the n-bit wordlength effects upon the operands involved in a single arithmetic operation. A separate function subroutine was developed for each operation and data type combination as shown in Table III. Each routine simulates n-bit wordlength effects according to the options of the user. The user states the options desired by a call to a SETNBIT routine to be subsequently discussed in detail. The various user options are stated in Figure 20. Options 2, 3, and 4 apply only to floating point computations. The following subsections will show how each option is carried out.

User specifications:

1) Total number of bits per word
2) Number of bits in exponent and number of bits for mantissa, in addition to the position of the implied decimal point with respect to the mantissa. The sum of the mantissa and exponent bits must be one less than the total number of bits specified in 1). The extra bit is for the sign.
3) Rounding or truncation effects for computations.
4) Single, double, or triple precision arithmetic accuracy.
5) Overflow messages print or suppression.

Figure 20 User Specifications (Options)

Option One. The first option indicates the total number of bits in the wordlength to be simulated. The SENBIT routine computes the maximum value that a fixed point word of n-bit wordlength could contain. The absolute value of

each fixed point result is compared to this maximum. If the maximum is exceeded, overflow has occurred. The value that overflowed is then replaced by the maximum positive value possible if the positive maximum was exceeded, or the maximum negative value possible if the negative maximum was exceeded.

Option Two. The second user option defines the data format to be used for floating point values. The different specifications include the number of mantissa and exponent bits along with implied decimal point with respect to the mantissa (either to its left or to its right). They are used by SETNBIT to compute the maximum and minimum positive floating point values representable for the n-bit word. If the absolute value of a floating point result exceeds these limits, an overflow or underflow condition has occurred. Underflow means the result represented was a smaller value than could be stored in an n-bit word with the given number of exponent bits. Upon detection of underflow, the value that underflowed is replaced with zero. Overflow detection is treated similar to fixed point overflow, except the maximum floating point values are used.

Also, for floating point values, the proper number of significant mantissa bits must be maintained. This is achieved by a right-shift (sign extended) followed by a circular left-shift of the floating point variable. For example, if 18 bits were specified for the mantissa, the rightmost 30 significant bits would be truncated, as shown in Figure 21.

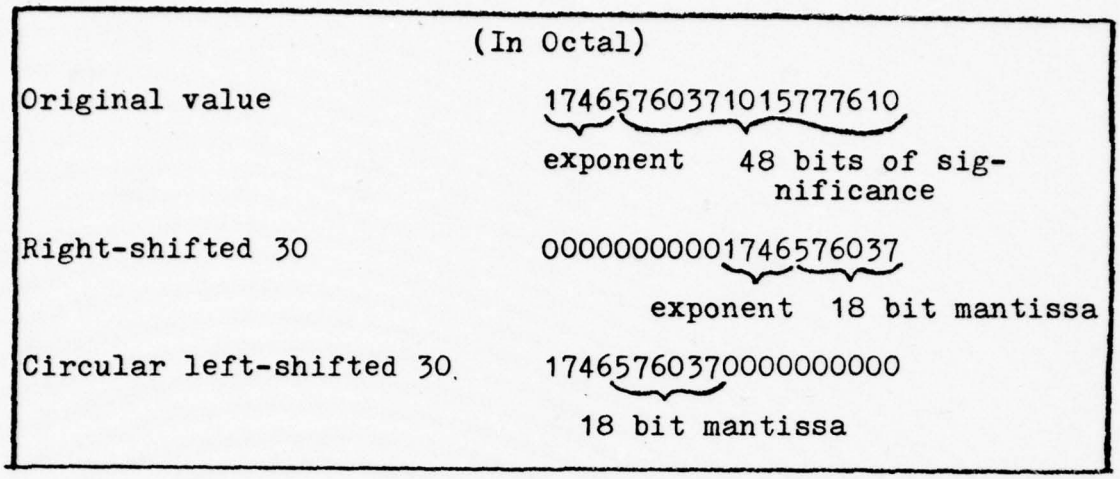


Figure 21 N-bit Simulation Of Mantissa

Option Three. The third user option is rounding or truncation effects. If truncation is specified no special action has to be taken, because the CDC 6600/CYBER 74 computers truncate by default. To round, however, a value equal to one half of the least significant bit (LSB) is added to the absolute value of the word. This is accomplished by right-shifting the number of mantissa bits specified; so that the least significant bit is in the second bit position from the right of the CDC 60-bit word (the 59th bit). Then the value one is added (using fixed point addition) to the quantity and the resulting quantity is right-shifted one bit. If the LSB-1 is a one, then the result would be rounded up; if it is not, it would have no effect. Figure 22 shows a case of a 15-bit mantissa in which the result is rounded up. After the one bit right-shift, the floating point value is left-shifted back to its original position.

A special condition may result from this technique of rounding. This condition occurs when the addition of one to

```
Original value:
00111110011010111111010111011100011010110001101000100000000000
After right-shift of 32 bits:
00000000000000000000000000000000001111100110101111110101110111
Plus
00000000000000000000000000000000000000000000000000000000000001
Sum equals:
00000000000000000000000000000000001111100110101111110101111100
After right-shift of 1 bit:
00000000000000000000000000000000000111110011010111111010111110
Circular left-shift of 33 bits (with rounded result):
00111110011010111111010111110000000000000000000000000000000000
```

Figure 22 Example Of Rounding

the right-shifted floating point value results in an increment of the exponent. The n-bit mantissa contains its maximum representation in these cases. Figure 23 shows a rounded value for a floating point value with a 15 bit mantissa. The rounded result, shown in (b) has no significant mantissa bits and would be treated from there on by the computer the same as a floating point zero. However, in a n-bit

Original value	(a)	17257777750031152437
		15 bit mantissa
Rounded value (incorrect)	(b)	17260000000000000000
Correct rounded value	(c)	17264000000000000000

Figure 23 Special Case Of Rounding

computer the round up of the value in (a) would result in an increment of the exponent and the mantissa would have a one in its most significant position, as shown in (c).

Option Four. The fourth user option is to specify single, double, or triple precision arithmetic accuracy. This option is used to simulate the effects of a whole complex equation being computed with multi-word precision, then storing the result as a single precision value. The preprocessor passes a flag which indicates if the n-bit subroutine computation will result in an intermeidate value or a final assignment. The SETNBIT routine computes the bit shifting needed to be done for the intermediate values.

A floating point 15-bit word with nine mantissa bits and the triple precision option would be handled as shown in Figure 24. In a triple word accumulator of a 15-bit computer, there would be 39 significant mantissa bits (i.e. 9 bits plus two additional 15-bit words used to extend the mantissa significance). The final value assigned would be handled just as any 15-bit single precision floating number, similar to Figure 21.

Original value	174657603710157776l5
Right-shift 9 bits	000174657603710157777
Left-shift 9 bits	174657603710157770000
	39 bit mantissa

Figure 24 Multiple Precision Accuracy

Option Five. The fifth option can be used to print or suppress overflow messages when the n-bit wordlength is

exceeded. A test is made in the overflow handling routine. If the message flag is off (set to zero), no message is printed. If the message flag is on (set to one), a message is printed indicating the line number and routine in which the overflow resulted from and the operation being performed at the time the condition was detected.

N-bit Simulation Sequence

The general sequence of events involved in simulating n-bit wordlength effects for each n-bit simulation routine is listed below:

1) Perform the arithmetic operation using the two input operands and storing the result. Input operands are assumed to have n-bit wordlength significance already.
2) If the rounding option (which applies only to floating point computations) is used, rounding effects are performed on the result.
3) Check the result for overflow. If overflow conditions exist, branch to overflow handler to substitute a value for the overflowed result and print an overflow message (if the message print option is on).
4) For floating point quantities, truncate the mantissa of the result to the number of bits specified for the n-bit floating point mantissa. If this is an intermediate result (indicated by the Final Flag input argument), the result's mantissa is truncated

to the precision specified (single, double, or triple).

5) Return to the calling routine.

SETNBIT. The subroutine SETNBIT computes boundary values based on the user options selected (Figure 20). The SETNBIT subroutine is required to be the first executable statement in each program unit. The user specifies the options desired as arguments to the SETNBIT subroutine. Different routines may execute with different options. Each routine's options are translated into seven key values which are localized to each routine in an array called KEY. The array KEY is used by n-bit simulation function subroutines to carry out the n-bit wordlength effects specified by the user. A description of each of the seven values of the KEY array and how each is computed is as follows.

KEY(1) represents the maximum fixed point value representable with n-bits. The maximum value is computed by setting the sign bit (left-most bit in a CDC 60-bit word), then circular left-shifting n, the number of bits in the wordlength, and subtracting one, as shown in Figure 25 (where n equals 15).

	(In Octal)
1. Set sign bit:	4000000000000000000
2. Circular left-shift 15 bits:	0000000000000040000
3. And subtract one:	0000000000000037777

Figure 25 Computation Of Maximum Fixed Point Values

KEY(2) represents the maximum floating point value possible for an n-bit floating point word with the specified floating point word characteristics (exponent/mantissa bit lengths and the implied decimal point position). This maximum value is computed by determining the maximum range for the n-bit word exponent. This exponent maximum is computed exactly in the same manner as KEY(1). The exponent range would be equal to one-half the maximum exponent representable. For a five-bit exponent and a ten-bit mantissa, the maximum representable would be (in binary) 11111 or 2^{+31}. The minimum exponent would be 00000 or 2^{-31} (assuming a bias of 10000). If the implied decimal point is to the left of the mantissa (making the whole mantissa a fractional value), the maximum exponent is decreased by the number of mantissa bits (ten for this example) in the n-bit word. If the implied decimal point is to the right, the maximum exponent is not affected.

KEY(3) represents the least significant floating point value representable with the specified mantissa, exponent, and implied decimal point position. Figure 26 shows this value for a 16-bit word with a five-bit exponent and ten-bit mantissa with the implied decimal point to the right. To compute this value, the fixed point representation of the mantissa, shown in Figure 26 is multiplied by two to the

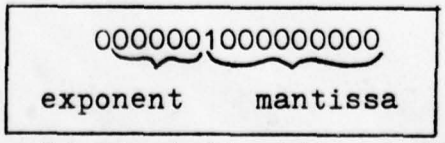


Figure 26 Example Of Least Significant Floating Point Value

maximum negative 31 exponent (-31 for a five-bit exponent). If the mantissa is a fractional representation for this example, the number of mantissa bits (ten) is subtracted from the maximum negative exponent (-31). So the least significant value in this case would be the fixed point mantissa value times $2^{(-31 - 10)}$ or 2^{-41}. This last value would be equivalent to .5 times 2^{-31}. Any value computed in the n-bit simulated algorithm with less significance would constitute an underflow for the n-bit word.

KEY(4) and KEY(5) simply contained the options indicated for rounding/truncation and message printing/suppression, respectively.

KEY(6) indicates the number of bits a mantissa must be right-shifted then left-shifted to retain the proper amount of mantissa significance, which would be 48 minus the number specified. Forty-eight is the number of bits in the mantissa of CDC's 60-bit word.

KEY(7) indicates the number of bits a mantissa must be shifted to retain single, double, or triple precision accuracy. This is computed by adding the number of mantissa bits to another full-wordlength (n-bits), for double precision, or to two wordlengths for triple precision. Then that result is subtracted from 48. For a 16-bit word with a nine-bit mantissa, the resulting quantity for single precision would be 39 or (48 - 9). For double precision, it would be 23 or (48-(9+16)). For triple precision, it would be 7 or (48-(9+16+16)).

Besides precomputing the values for the KEY array, SETNBIT performs validation checks on the options to verify they are consistent and can be simulated using the CDC 60-bit wordlength. Failure of a validation check results in program termination.

The Overall N-bit Simulation Tool

To use the n-bit simulation tool, a user must first comply with the restrictions which are described in Appendix B. Two major restrictions are to debug the FORTRAN algorithm from all syntax errors prior to execution, and to assign all external values of the program (such as, through a read) to themselves. For the variable L5, the statement L5 = L5 would reduce the value in L5 to its n-bit wordlength significance. One additional restriction is that the first executable statement of each program routine must be a call to the SETNBIT subroutine containing the user's options, and the array variable KEY must be dimensioned for seven items. The SETNBIT subroutine call allows the user to specify different n-bit wordlength characteristics for different subprograms within the same set of programs.

The preprocessor will read the FORTRAN algorithm and transform each arithmetic statement into a series of function subroutine calls. The preprocessed program would then be compiled and loaded with the n-bit simulation function subroutines and executed. During program execution, the n-bit function subroutines would perform the arithmetic

operations and assignments while simulating the effects of n-bit wordlength and other user options.

Appendix B contains the User's Guide to the n-bit simulation tool and Appendix C contains a Detailed Description of the preprocessor and the n-bit simulation subroutines. The next chapter analyzes the design of the preprocessor in further detail and describes the modules contained within it.

IV. Structured Design Of The Preprocessor

This chapter describes the final design of the n-bit simulation preprocessor which resulted from the application of structured design techniques. It also discusses how applicable structured design techniques were found to be for a small system design (i.e. the preprocessor) of approximately 1000 source statements.

Structured design is "the art of designing the components of a system and the interrelationship between those components in the best possible way" (REF 9: 7). Basically, it is a form of top-down design. One structured design method is transform analysis. Transform analysis is a strategy that derives initial structure designs which are usually quite good and require only modest restructuring to arrive at the final design. The quality of a design can be measured in terms of its reliability (the number of "bugs" encountered in the program), maintainability (the amount of effort required to fix "bugs" in the program), and modifiability (the cost of changing or extending the program).

Data Flow Description

Using the transform analysis approach, a data flow diagram of the preprocessor was developed, as shown in Figure 27. The preprocessor inputs a FORTRAN statement, then scans the statement for a sequence of characters which could represent an object of an arithmetic statement. An object refers to the variable to which the result of arithmetic

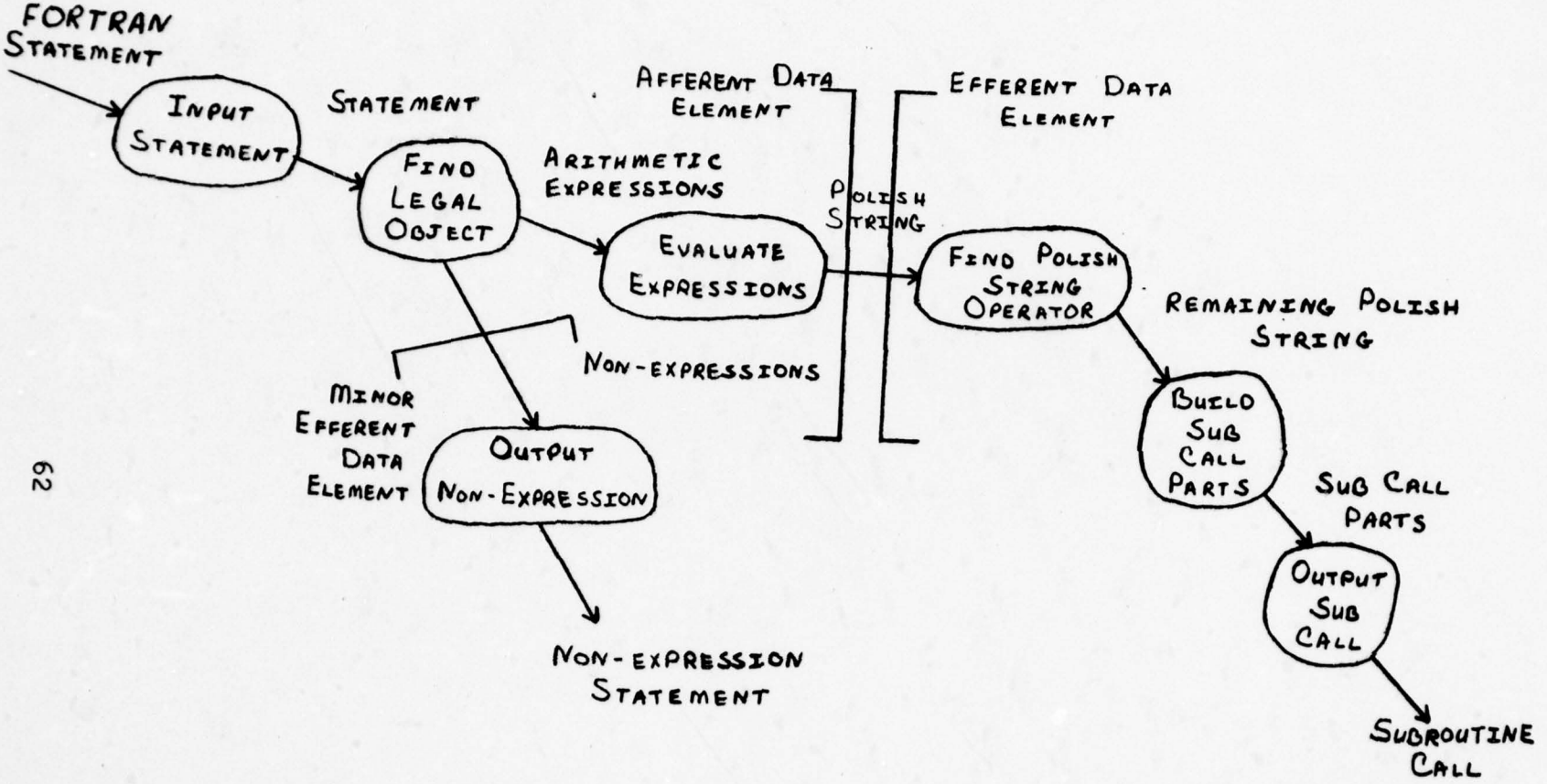


Figure 27 Preprocessor Data Flow Diagram

equation is assigned; where an arithmetic statement is of the general form "object = arithmetic expression". Any statement not having a legal object is regarded as a non-expression and is output in its original form to the n-bit simulated program file.

Upon detection of a legal object, the scan continues to the right side of the '=' in the arithmetic statement to evaluate the expression and convert the statement into its equivalent Polish Notation representation.

The right half of the data flow diagram describes the process in which the Polish Notation string is scanned for operators. Each operator found will result in the building of a function call. To accomplish this, the operands which are being operated upon are found in the Polish string, then merged with the function routine name and other data required to build the function call (such as routine name, line number, and the reference to the KEY array).

The afferent and efferent data elements are defined in the data flow design also. "Afferent data elements are those high-level elements of the data which are furthest removed from the physical input, yet still constitute inputs to the system. Efferent data elements are those furthest removed from the physical outputs which may still be regarded as outgoing." (REF 9: 262). In the case of the preprocessor, the Polish Notation string representation of each arithmetic statement constitutes the afferent as well

as the efferent data elements. All processes to the left of the afferent data element are dedicated towards building the Polish string from the input FORTRAN statement. All those to the right are intent upon converting the Polish string into a series of subroutine calls.

In developing a program structure from the data flow diagram, transform analysis states that major afferent and efferent data elements should be handled by separate modules and that these modules should be immediately subordinate to the main module of the preprocessor. The basic module structure which should be developed from this data flow diagram is shown in Figure 28. The fully factored structure which resulted for the final design is shown in Figure 29. The module definitions for the resulting preprocessor design and the module interfaces are contained in Appendix C. The following section explains the differences between the fully factored structure of Figure 29 and the structure that would be expected to follow from Figure 28.

Departures

Ideally, with the transform analysis strategy, every module of the afferent branch should have only afferent or transform subordinates, where afferent characteristics imply that data is floating inward (actually upward) with respect to the system. Transform modules transform the input data closer to its most abstract input representation

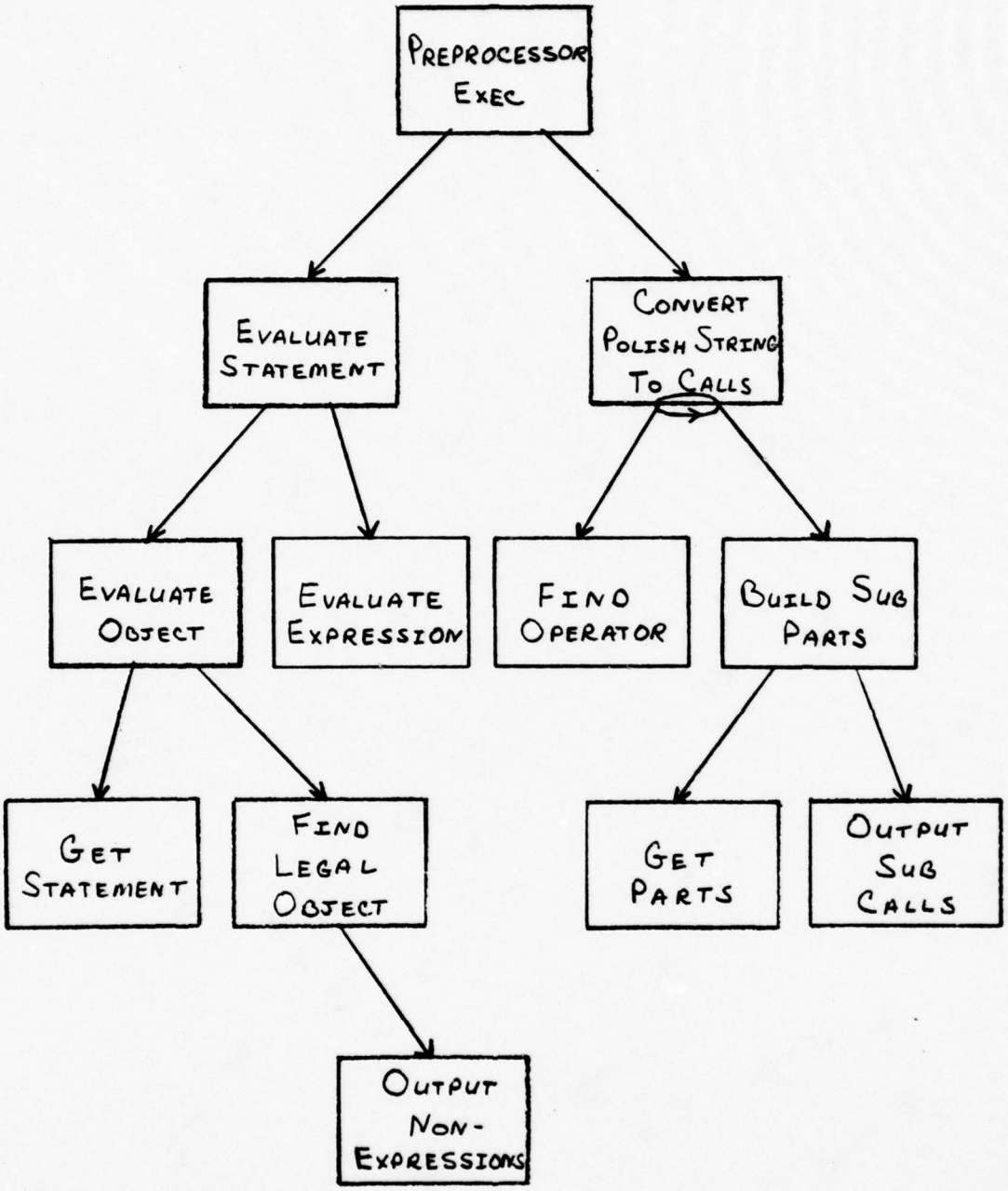


Figure 28 Initial Program Structure For The Preprocessor Data Flow Diagram

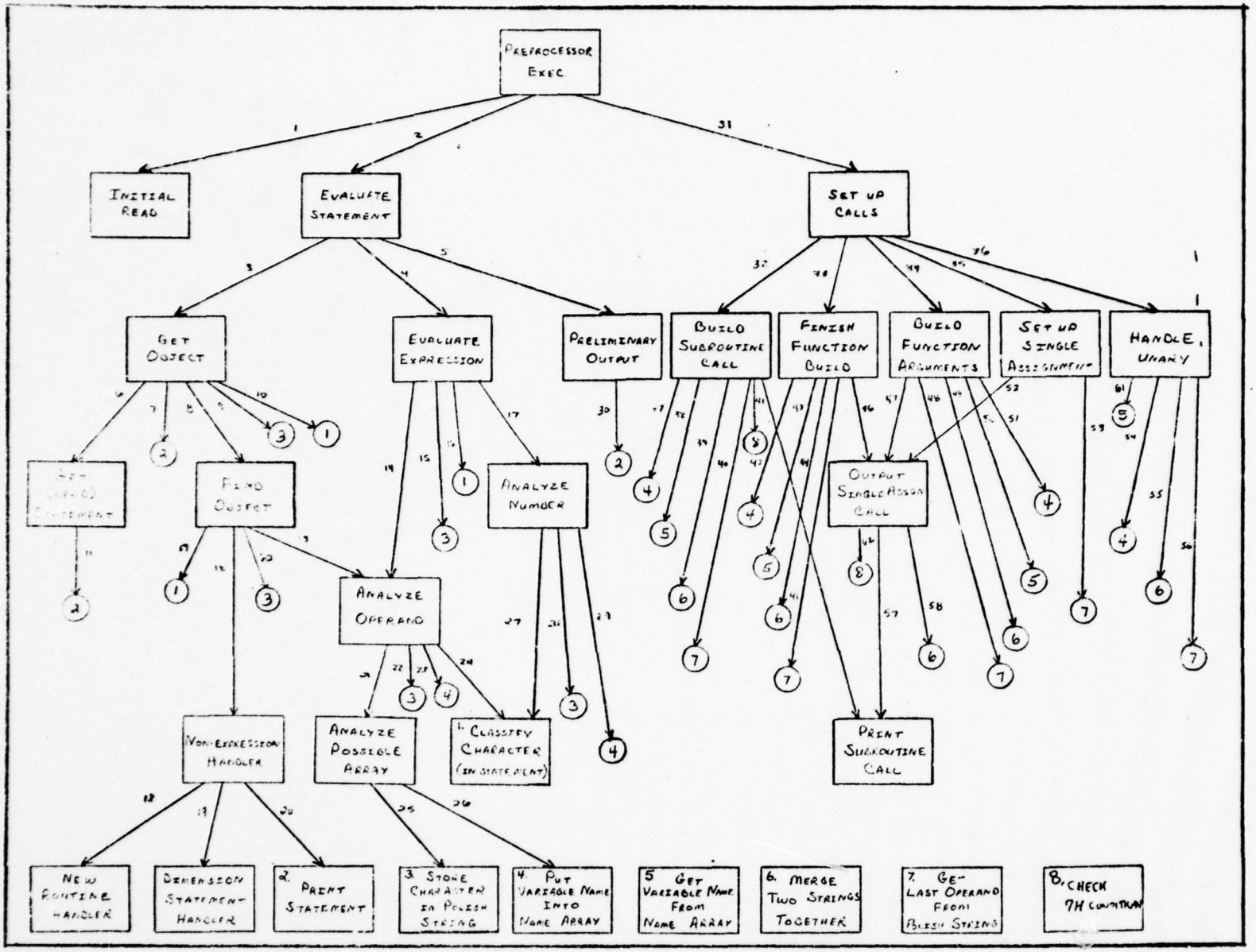


Figure 29 Final Program Strucure For Preprocessor

for the system. Efferent (data flowing outward) modules are expected to have only efferent and transform subordinates, where in this case the transform modules transform data closer to its output form. Anything contrary to a consistent data flow (either inward or outward) is termed a departure. The departures of the resulting structure of Figure 29 are discussed in the following paragraphs.

Two of the departures are for two minor efferent modules which appear inside the afferent branch of the preprocessor structure. These are the PRINT-STATEMENT and PRELIMINARY-OUTPUT modules. The PRINT-STATEMENT module is called when any module within the afferent branch detects a non-arithmetic statement. Only arithmetic statements are converted into Polish Notation and subsequently transformed into subroutine calls. All other statements are passed along to the n-bit simulated program file unchanged.

The PRELIMINARY-OUTPUT module is performed once an arithmetic statement has been converted into its Polish Notation equivalent representation. This routine sets up prints to the output file for a comment containing the original arithmetic statement and the preliminary output that is required for labels and IF statements. Perhaps this module should have been included under the efferent branch module structure, but it requires the original input statement. No module in the efferent branch requires the original statement, while almost every afferent branch module does. In order to prevent the original input of the

system from being passed unnecessarily into the efferent branch side, the PRELIMINARY-OUTPUT module was included as the last step of the afferent branch.

The INITIAL-READ module which is directly subordinate to the executive module represents another departure. This module is used to read the first program card image. All succeeding reads are performed by the GET-STATEMENT module, which operates on a 'look-ahead' basis, in order to detect statements that are continued on other cards. The INITIAL-READ module does the initial read so the GET-STATEMENT module does not have to repeatedly test for the initial read on all cards in the program.

The other departures of the final structure in Figure 29 appear in the efferent branch. The module called FINDING-THE-OPERATION in Figure 28 was such a short process that it was merged into the SET-UP-CALLS module. Also, the GET-PARTS module was merged with its subordinate (BUILD-SUB-CALLS) because they were so strongly data coupled. Data coupling measures the amount of data passed as arguments between modules as subroutine parameters. In this case, the GET-PARTS module was very simple and the data coupling between it and BUILD-SUB-CALLS was high enough that a separate module was not justified.

The Effects Of Structured Design

One of the goals of this thesis was to determine how applicable structured design concepts were to small programs.

The importance of structured design in large programs has received considerable publicity, but the programs of less than 5,000 source statements are more common place in the programing world. The n-bit simulation preprocessor is an example of a small program design.

Part of the evaluation of the resulting structured design is based on its comparison to a design developed for the preprocessor before the structured design approach was begun. The previous, or first, program design did not take a formalized approach to structuring the design. It was based upon a flowchart of the preprocessor concept. Many of the modules resulted from blocking off portions of the flowchart. The first design's program modules were fairly intricate and hard to understand. The structured design approach emphasizes that the structure of the program should resemble the structure of the problem, which the first design certainly did not.

The first design consisted of 16 modules that were interdependent on each other. Modules were not designed to be independent. Modules split up functional tasks. When a module called another, certain assumptions were being made. As a result, a program modification to one module may have indirectly affected another. Program modifications could not be performed without analyzing many modules to evaluate the impact of the change. The whole design was essentially tailored to solving this problem, without much consideration to how future changes could be incorporated.

It is probable that as the program was modified it's complexity would go up and it's maintainability down. Though the first design was never carried out to completion, it was clearly inferior to the final structured design.

The structured design consisted of 30 modules. It seemed to be simpler, more straight-forward, and more general approach to the problem. These modules were generally much more functional in nature. In fact, ten of the 30 modules were so generally defined that each was used by three to eight calling modules. A new maintenance programer could more readily understand the design since the structured design was more functionally broken down.

Another aspect of the structured design was that variables were initialized at the lowest possible level. In the first design, nearly all initializations were performed by one module. Initialization in the first module where the variable is needed is more desirable, because when that module is modified it is possible an initialization may have to change. Since the modules are so far removed, the change might easily be overlooked.

Data coupling of modules was used for the preprocessor interface in all but four cases. In three of the cases, COMMON coupling was chosen over data coupling because in each case only two modules needed to share the data and these modules were separated by several module levels. COMMON coupling of the two modules was believed better than passing the data along as arguments through many modules

that had no need for the information. By COMMON coupling the two modules, it prevented other modules from having needless access to the data.

The other exception to data coupling involved the passing of a control flag as a subroutine argument. This flag was passed down to the BUILD-SUBROUTINE-CALLS module. It indicated the subroutine call being built was the final assignment for the arithmetic statement, and no temporary variable would need to be created for storing the result.

No other control flags were used specifically. Alternate returns from subroutines were used in place of control flags. These alternate returns were used for end-of-file conditions and for signaling the detection of an illegal arithmetic statement. The alternate returns for an illegal arithmetic statement were used to avoid further syntax analysis of the statement. Non-arithmetic statements do not need to be converted to Polish Notation form. Myers (REF 6: 51), however, does not regard this usage of alternate returns as control flags since the return indicates to the superordinate module that the function failed. Myers regards this type of return code as data since it does not tell the calling module what to do, but says 'I've failed' and the calling module can respond however it wants.

One other design improvement over the first design, resulting from the structured design approach, was better scope of effect/scope of control. Scope of effect/scope of control essentially rationalizes that all modules effected

by a decision should be subordinate to (beneath) the module making the decision. A consequence of disregarding the scope of effect/scope of control rule is that decisions are repeated as control flags are passed back to the superordinate to be retested.

The preprocessor design kept the scope of effect for a decision within the scope of control of the module making the decision. Under the first design of the preprocessor programs, the detection of an illegal arithmetic statement resulted in passing a control flag back to the superordinate module which output the statement for that subordinate's decision and several other subordinate modules which detected syntax errors. In the structured design, the statement was output by the module detecting the illegal syntax.

Testing The Quality Of The Design

An important test for rating the quality of a design is the ease with which modifications can be made to the existing program. For one proposed change to the program, the design was found quite adaptable. This change is described in the following subsection. The change is explained, then the modules and interfaces affected are listed. In addition, the impact of the change upon the quality of the present program is evaluated.

One change was to replace all subroutine calls built by the efferent branch with in-line code that would do the n-bit simulation effects. The in-line code would be tailored

to the user options in effect for that particular routine. This would mean fewer decisions would be made (such as testing if the rounding effects option is on). Also subroutine linkage time would be saved since the n-bit simulation code was no longer performed by a subroutine. This would result in improved execution time for the n-bit simulation of an algorithm. This proposed change is described completely in Appendix D, as a future enhancement to the system.

Basically, this change would require modifications to three of the current modules: BUILD-SUBROUTINE-CALLS, OUTPUT-SINGLE-ASSIGNMENT, and NON-EXPRESSION-HANDLER. Also two new modules would need to be developed. The modifications to BUILD-SUBROUTINE-CALLS and OUTPUT-SINGLE-ASSIGNMENT would simply be the replacement of one block of statements within each one. This block of statements presently merges portions of the function subroutine being built. The new code introduced within these routines would merge these same parts already present in a different way and call a new module which would insert these parts into standard positions to build the appropriate series of in-line statements for n-bit simulation.

Four additional interfaces would be introduced into the present code; three of them to join the new modules to their superordinate modules and the other would interface the two new modules through the use of COMMON coupling. No present interfaces would be changed and the strength of each module modified would not be decreased. The changed module,

NON-EXPRESSION-HANDLER, presently detects key statements and calls the appropriate handling routine. The CALL SETNBIT subroutine reference could be detected using the same technique, then a new module to handle the statement could be called. This new module would analyze the arguments of the SETNBIT subroutine and compute the key values normally computed by the SETNBIT subroutine of the present system, then pass the keys as data through an interface to the other new module created for this enhancement.

All changes for this program enhancement could be accomplished quickly, easily with firm confidence that the system would be just as functional and easy to maintain as before.

Conclusion

In conclusion, structured design techniques were found to be valuable in the creation of a better design. The final design had many significant advantages over the previous design which relied on no actual methodology. The structured design resulted in a simpler system since its structure resembled the problem structure and the modules were functional in nature. Simplicity is an important design objective. A program that is simple and easier to understand has a more positive effect on the future maintenance and modification costs of the program.

Even though the structured design analysis took considerably more time, it was felt the time would be regained

in the succeeding stages of the preprocessor system life-cycle. The integration of the preprocessor modules and the debugging went quickly and smoothly once the design was coded. The testing and the results obtained are discussed in the next chapter.

Based on the preprocessor program, the structured design techniques were found to be very applicable to this small program design. Perhaps the cost of poor designing is not as great for a small program as for a large program, but small programs comprise a large percentage of the programing systems around the world. Therefore, the overall impact of structured design could be even greater than applying it to only large systems. The next chapter will review the result from tests performed upon the preprocessor developed from the structured design and the other parts of the n-bit simulation tool.

V. Testing And Results

This chapter reviews the tests performed on the two portions of the n-bit simulation tool, the preprocessor and the n-bit simulation subroutine. It then examines some of the results obtained from two examples of FORTRAN algorithms to demonstrate some of the effects obtainable through the n-bit simulation tool.

Preprocessor

The objective of the preprocessor is to translate a given FORTRAN algorithm into its equivalent n-bit simulation form. This process converts all syntactically correct arithmetic statements into a sequence of one or more function subroutine calls.

A test program including various forms of each FORTRAN statement type was developed to verify that non-arithmetic statements would pass through the preprocessor unaffected and that virtually any form of an arithmetic statement would be correctly translated into the intended sequence of function subroutine calls. This sample program is shown in Figure 30. The n-bit simulated version of this program produced by the preprocessor is shown in Figure 31.

All statement types were processed correctly by the preprocessor. The sequence of function calls for each arithmetic statement reflected the proper precedence order of evaluation for the operators and operands involved. (A comment statement containing the original statement preceded

```
      PROGRAM PREPRS(INPUT,OUTPUT,TAPE1=OUTPUT)                          000100
      IMPLICIT INTEGER  (A-Z)                                            000110
C     IMPLICIT INTEGER IS NOT ALLOWED IN THE N-BIT SIMULATOR             000120
C       NOR ARE INTEGER OR REAL VARIABLE DECLARATIONS, VARIABLE          000130
C       TYPE SHOULD USE DEFAULT VALUES I-N FOR INTEGER,AND               000140
C          A-H,O-Z FOR REALS                                             000150
      DIMENSION C(20),D(15,2),K(15),KEY(7)                               000160
      COMMON A,B,DDDD                                                    000170
      INTEGER A2,A4                                                      000180
      EQUIVALENCE (C5,D5),(I1,D6)                                        000190
      DATA  C/20*1.5/,BLNK/1H /                                          000200
C     THIS IS   ONLY A TEST PROGRAM TO EXAMINE THE PREPROCESSORS         000210
C       HANDLING OF THE VARIOUS STATEMENT TYPES                          000220
C       A " CALL SETNBIT" MUST BE THE FIRST EXECUTABLE STATEMENT WITHIN   000224
C        EACH ROUTINE AND KEY(7) MUST BE DIMENSIONED FOR EACH ROUTINE    000225
C        IN THE N-BIT SIMULATED PROGRAM                                  000226
      CALL SETNBIT(24,1,1,1,17,6,0,KEY)                                  000230
      READ *,C5,I1                                                       000240
C       ALL EXTERNAL VALUES COMING INTO A N-BIT SIMULATED ROUTINE        000241
C       MUST BE SET EQUAL TO THEMSELVES-IN ORDER TO GIVE EACH ITS N-BIT  000242
C         SIGNIFICANCE                                                   000243
      C5=C5                                                              000250
      I1=I1                                                              000260
C     EVERY EXTERNAL VARIABLE MUST BE ASSIGNED TO ITSELF TO              000270
C       N-BIT SIMULATE ITS ACCURACY                                      000280
      READ (1,3)  (K(L),L=1,15)                                          000290
    3 FORMAT(15F9.3)                                                     000300
C       THE ARRAY K (EXTERNAL VALUES) MUST BE SET EQUAL TO ITSELF TO     000305
C        GIVE IT ITS N-BIT WORDLENGTH SIGNIFICANCE                       000306
      DO 4 L=1,15                                                        000310
      K(L)=K(L)                                                          000320
    4 CONTINUE                                                           000330
      NN=(23+3*(PRV-1000))                                               000340
      DO 5 M=1,15                                                        000350
      C(M)=K(M)*2                                                        000360
    5 CONTINUE                                                           000370
      PRINT *," INPUT "                                                  000380
      PRINT *,K                                                          000390
      A1=K(1)+78924/K(2)                                                 000400
      CALL SUB1(25.,C,A1)                                                000410
      A=1**2/3+I+C(12)                                                   000420
      CALL MANIPUL(A,B,X,I),RETURNS(13)                                  000430
      DEFIND 1                                                           000440
      ASSIGN 13 TO ILABEL                                                000450
      DO 7 L=1,15                                                        000460
      C(L)=C(L,1)+25.37                                                  000470
    7 CONTINUE                                                           000480
      GO TO (13,15),AILABEL                                              000490
   10 IF(I1.EQ.0)      GO TO 9                                           000500
      WRITE (1,19)  A1,(C(NN),NN=1,20,2)                                 000510
   19  FORMAT(2X,=A10)                                                   000520
      GO TO 13                                                           000530
    9 PRINT *," STARTED LAST PART"                                       000540
   13 A1=C(20)*C(19)+K(4)**(-1.7)                                        000550
   15 A2=C(1)*(A1+K(4))-                                                 000560
     *      (A1-5)+1.000                                                000570
      STOP "  PREPROCESS TEST COMPLETE"                                  000580
      END                                                                000590
      SUBROUTINE SUB1(C1,C,A1)                                           000600
      COMMON A,B,DD                                                      000610
      DIMENSION C(1),KEY(7)                                              000620
      CALL SETNBIT(24,1,1,1,17,6,0,KEY)                                  000630
```

Figure 30 Sample Program

```
      C1=C(I1)/3.1423F678
      X=3.5
      RESULT=SIN(X)*A1**3.5
      NEXT=IFUNC(A1,X,C1*15.3/A1)
      I1=-I1+A1-C(I1)-4
      IF(I1.EQ.4)      GO TO 17777
      IF(I1.EQ.3)    A1=NEXT**(-2)+I1**(-2)
17777 RESULT=RESULT+0.5
      M=I1+40
      SUM=0.
      DO 19 K=1,M
      C(M)=C(M)+RESULT*3.1454
      SUM=C(M)+SUM
      IF(SUM.GT.2000.)     SUM=SUM/A1-100
   19 CONTINUE
      RETURN
      END
      FUNCTION IFUNC(A1,A2,A3)
      DIMENSION KEY(7)
      CALL SETNBIT(2+,1,1,1,17,6,0,KEY)
      IFUNC=0
      IF(A1.EQ.SHIFT(A2,5).AND.A2.EQ.5.9)
     * IFUNC=1
      RETURN
      END
      SUBROUTINE MANIPUL(A,B,C,K),RETURNS(N2)
      COMMON/NEXT/ NOTHING
      DIMENSION KEY(7)
      DIMENSION A(15),                         LARRAY(29),        I(5,3,3),
     *       C1(4),           C2(6),            MARRAY(23,2)
      LOGICAL L2
      L2=.TRUE.
      IF(C1(2).EQ.K)        L2=.FALSE.
      I(1,1,1)=25+5-7*K*LARRAY(K)*MARRAY(K,1)
   17 IF((LARRAY(7)**).EQ.LARRAY(7+4))      LARRAY(7)=0.1
      RETURN
      END
      INTEGER FUNCTION IFUNC2(I1)
      DIMENSION   KEY(7)
      CALL SETNBIT(2+,1,1,1,17,6,1,KEY)
      IFUNC2=FUNC3(I1**.+I1,I1,5)
      IF(IFUNC2.GT.  00)      IFUNC2=0
      RETURN
      END
      REAL FUNCTION FUNC3(I,I2,I3)
      DIMENSION KEY(7)
      CALL SETNBIT(12,1,3,1,7,5,1,KEY)
      FUNC3=I/I2/I3 (I2+I3*(4 . +I2))
      IF(FUNC3.LT.7.)    GO TO 40
      FUNC3=FUNC3*FUNC3
   40 RETURN
      END
      SUBROUTINE ERRORS(DUMMY,DUM1,DUM2)
      DIMENSION A(27),B(1,33)
      A1=B+C+
      RESULT=A=3
      [illegible]
      [illegible]=123.[illegible]
      RESULT=F[illegible](3,E,B*3,3+Y*TOTAL,F,G,H,SIN(X),I)
      [illegible]=K+L/3.[illegible]
      [illegible]=(4X+3**(4 . +.A4))
      [illegible]=123.[illegible]127.5677
      A(2,3)=[illegible]/A/
      [illegible]+K+1+2+1+K-[illegible]-1)=LO.G+ARRAY+ARGUMENT
      RETURN
      END
```

Figure 30 (Continued)

```
      PROGRAM PREPRSS(INPUT,OUTPUT,TAPE1=OUTPUT)                        000100
      IMPLICIT INTEGER  (A-Z)                                           000110
C     IMPLICIT INTEGER IS NOT ALLOWED IN THE N-BIT SIMULATOR            000120
C      NOR ARE INTEGER OR REAL VARIABLE DECLARATIONS, VARIABLE          000130
C      TYPE SHOULD USE DEFAULT VALUES I-N FOR INTEGER,AND               000140
C         A-H,O-Z FOR REALS                                             000150
      DIMENSION C(20),D(15,2),K(15),KEY(7)                              000160
      COMMON A,B,DDDD                                                   000170
      INTEGER A2,A4                                                     000180
      EQUIVALENCE (C5,D5),(I1,D6)                                       000190
      DATA  C/20*1.7/,BLNK/10H          /                               000200
C     THIS IS   ONLY A TEST PROGRAM TO EXAMINE THE PREPROCESSORS        000210
C      HANDLING OF THE VARIOUS STATEMENT TYPES                          000220
C      A " CALL SETNBIT" MUST BE THE FIRST EXECUTABLE STATEMENT WITHIN  000230
C       EACH ROUTINE AND KEY(7) MUST BE DIMENSIONED FOR EACH ROUTINE    000240
C       IN THE N-BIT SIMULATED PROGRAM                                  000250
      CALL SETNBIT(2+,1,1,1,1?,8,4,KEY)                                 000260
      READ *,C5,I1                                                      000270
C      ALL EXTERNAL VALUES COMING INTO A N-BIT SIMULATED ROUTINE        000280
C       MUST BE SET EQUAL TO THEMSELVES-IN ORDER TO GIVE EACH ITS N-BIT 000290
C        SIGNIFICANCE                                                   000300
C      C5=C5                                                            000310
       C5=RASGN(C5,7HPREPRSS, 32,0,KEY)                                 000320
C      I1=I1                                                            000330
       I1=IASGN(I1,7HPREPRSS, 23,0,KEY)                                 000340
C     EVERY EXTERNAL VARIABLE MUST BE ASSIGNED TO ITSELF TO             000350
C       N-BIT SIMULATE ITS ACCURACY                                     000360
      READ (1,3)  (K(L),L=1,15)                                         000370
    3 FORMAT(15F8.3)                                                    000380
C       THE ARRAY K (EXTERNAL VALUES) MUST BE SET EQUAL TO ITSELF TO    000390
C        GIVE IT ITS N-BIT WORDLENGTH SIGNIFICANCE                      000400
      DO 4 L=1,15                                                       000410
C     K(L)=K(L)                                                         000420
      K(L)=IASGN(K(L),7HPREPRSS, 31,0,KEY)                              000430
      CONTINUE                                                          000440
C     M11=(23+3*((23/-1)*0))                                            000450
      ATTTAA=RSHNS(20/,1 0.,7HPREPRSS, 33,0,KEY)                        000460
      ATTTAA=RSHFT(3,ATTTAA,7HPREPRSS, 33,0,KEY)                        000470
      M44=RLSHT(23,ATTTAA,7HPREPRSS, 33,1,KEY)                          000480
      DO 5 M=1,15                                                       000490
C     C(I)=K(M)**2                                                      000500
      C(1)=IEMPY(K(I),2,7HPREPRSS, 35,1,KEY)                            000510
    5 CONTINUE                                                          000520
      PRINT *," INPUT "                                                 000530
      PRINT *,K                                                         000540
C     A1=K(1)*Z5524/K(2)                                                000550
      TTTTTT=IIDV7(7777?4,K(2),7HPREPRSS, 39,0,KEY)                     000560
      A1=IIAD7(K(1),TTTTAA,7HPREPRSS, 39,1,KEY)                         000570
      CALL SUB1(C7,,C,A1)                                               000580
C     A=A**2/3*I+C(12)                                                  000590
      ATTTTA=RDEX8(A,2,7HPREPRSS, -1,0,KEY)                             000600
      ATTTAA=RSHV7(ATTTAA,2,7HPREPRSS, -1,0,KEY)                        000610
      ATTTTA=RSHFT(ATTTAA,1,7HPREPRSS, -1,1,KEY)                        000620
      A=RADD(ATTTTA,C(12),7HPREPRSS, -1,1,KEY)                          000630
      CALL SUB1(A,,C,Y,I),RETURN(13)                                    000640
      PRINT 1                                                           000650
      ASSIGN 13 TO ILABEL                                               000660
      DO 7 L=1,15                                                       000670
    7 C(L)=C(L,1)**2 .37                                                000680
      C(L)=RADD(C(L,1).2*.37,7HPREPRSS, 44,1,KEY)                       000690
      CONTINUE                                                          000700
      GO TO (13,1 ),ILABEL                                              000710
```

Figure 31 N-bit Simulated Version Of Sample Program

```
30    IF(I1.EQ.0)      GO TO 9                                          000720
      WRITE (1,19)  A1,(C(NN),NN=1,20,2)                                000730
19     FORMAT(2X,=A1))                                                  000740
      GO TO 13                                                         000750
9     PRINT *," STARTED LAST PART"                                     000760
13    CONTINUE                                                         000770
C     A1=C(20)*C(19)*K(4)**(-1.7)                                      000780
      RTTTTAA=RRMPY(C(20),C(19),7HPREPRSS, 54,0,KEY)                   000790
      RTTTTAB=R1EXP(K(4),-1.7,7HPREPRSS, 54,0,KEY)                     000800
      A1=RRMPY(RTTTTAA,RTTTTAB,7HPREPRSS, 54,1,KEY)                    000810
15    CONTINUE                                                         000820
C     A2=C(1)*(A1+K(4))*                                               000830
C           (A1-5)+1.000                                               000840
      RTTTTAA=R2ADD(A1,K(4),7HPREPRSS, 56,0,KEY)                       000850
      RTTTTAB=RRMPY(C(1),RTTTTAA,7HPREPRSS, 56,0,KEY)                  000860
      RTTTTAC=R2MNS(A1,5,7HPREPRSS, 56,0,KEY)                          000870
      RTTTTAD=RRMPY(RTTTTAB,RTTTTAC,7HPREPRSS, 56,0,KEY)               000880
      A2=R2ADD(RTTTTAD,1.000,7HPREPRSS, 56,1,KEY)                      000890
      STOP "  PREPROCESS TEST COMPLETE"                                000900
      END)                                                             000910
      SUBROUTINE SUB1(I1,C,A1)                                         000920
      COMMON A,B,DDD                                                   000930
      DIMENSION C(1),KEY(7)                                            000940
      CALL SETVPIT(24,1,1,1,17,6,0,KEY)                                000950
C     C1=C(I1)/3.14285678                                              000960
      C1=RRDVD(C(I1),3.14285678,7HSUB1    ,  5,1,KEY)                  000970
C     X=1.5                                                            000980
      X=RASGN(1.5,7HSUB1    ,  6,0,KEY)                                000990
C     RESULT=SIN(X)*A1**3.5                                            001000
      RTTTTAA=RASGN(X,7HSUB1    ,  7,0,KEY)                            001010
      RTTTTAB=RASGN(SIN(RTTTTAA),7HSUB1    ,  7,0,KEY)                 001020
      RTTTTAC=RREXP(A1,7.5,7HSUB1    ,  7,0,KEY)                       001030
      RESULT=RRMPY(RTTTTAB,RTTTTAC,7HSUB1    ,  7,1,KEY)               001040
C     NEXT=IFUNC(11,X,C1*15.3/A1)                                      001050
      RTTTTAA=RASGN(X,7HSUB1    ,  8,0,KEY)                            001060
      RTTTTAB=RASGN(A1,7HSUB1    ,  8,0,KEY)                           001070
      RTTTTAC=RRMPY(C1,15.3,7HSUB1    ,  8,0,KEY)                      001080
      RTTTTAD=RRDVD(RTTTTAC,A1,7HSUB1    ,  8,0,KEY)                   001090
      NEXT=IASGN(IFUNC(RTTTTAB,RTTTTAA,RTTTTAD),7HSUB81    ,  8,1,KEY)  001100
C     I1=-I1+A1-C(I1)-4                                                001110
      RTTTTAA=R1ADD(-I1,A1,7HSUB1    ,  9,0,KEY)                       001120
      RTTTTAB=R2MNS(RTTTTAA,C(I1),7HSUB1    ,  9,1,KEY)                001130
      I1=R2MNS(RTTTTAB,4,7HSUB1    ,  9,1,KEY)                         001140
      IF(I1.EQ.6)      GO TO 17777                                     001150
C     IF(I1.EQ.3)    A1=NEXT**(-2)*I1**(-2)                            001160
      IF(I1.EQ.3)                                                      001170
     *GO TO 7911                                                       001180
      GO TO 7912                                                       001190
 7911 CONTINUE                                                         001200
      RTTTTAA=R1EXP(NEXT,-2,7HSUB1    , 11,0,KEY)                      001210
      RTTTTAB=R1EXP(I1,-2,7HSUB1    , 11,0,KEY)                        001220
      A1=R1ADD(RTTTTAA,RTTTTAB,7HSUB1    , 11,1,KEY)                   001230
 7912 CONTINUE                                                         001240
17777 CONTINUE                                                         001250
C     RESULT=RESULT+1.5                                                001260
      RESULT=R2ADD(RESULT,1.5,7HSUB1    , 12,1,KEY)                    001270
C     I=I+2                                                            001280
      I=I1ADD(I1,10,7HSUB1    , 13,1,KEY)                              001290
C     SUM=1.                                                           001300
      SUM=RASGN(1.,7HSUB1    , 14,0,KEY)                               001310
      DO 19 K=1,4                                                      001320
C     C(4)=C(4)*RESULT+3.1494                                          001330
      RTTTTAA=RRMPY(RESULT,3.1[illegible],7HSUB1    , 16,0,KEY)        001340
      C(4)=R1ADD(C(4),RTTTTAA,7HSUB1    , 16,1,KEY)                    001350
C     SUM=C(4)+SUM                                                     001360
      SUM=R2ADD(C(4),SUM,7HSUB1    , 17,1,KEY)                         001370
```

Figure 31 (Continued)

```
C      IF(SUM.GT.2000.)     SUM=SUM/A1-100                                    001380
       IF(SUM.GT.2000.)                                                       001390
      *GO TO 7913                                                             001400
       GO TO 7914                                                            001410
 7913 CONTINUE                                                               001420
       RTTTTAA=RRDVD(SUM,A1,7HSUB1    , 18,C,KEY)                            001430
       SUM=R2MNS(RTTTTAA,100,7HSUB1    , 18,1,KEY)                           001440
 7914 CONTINUE                                                               001450
 19   CONTINUE                                                               001460
       RETURN                                                                001470
       END                                                                   001480
       FUNCTION IFUNC(A1,A2,A3)                                              001490
       DIMENSION KEY(7)                                                      001500
       CALL SETNBIT(24,1,1,1,17,6,0,KEY)                                     001510
C      IFUNC=0                                                               001520
       IFUNC=IASGN(0,7HIFUNC   ,  4,0,KEY)                                   001530
C      IF(A1.EQ.SHIFT(A2,5).AND.A2.EQ.5.9)                                   001540
C     * IFUNC=1                                                              001550
       IF(A1.EQ.SHIFT(A2,5).AND.A2.EQ.5.9)                                   001560
      *GO TO 7911                                                             001570
       GO TO 7912                                                            001580
 7911 CONTINUE                                                               001590
       IFUNC=IASGN(1,7HIFUNC   ,  6,0,KEY)                                   001600
 7912 CONTINUE                                                               001610
       RETURN                                                                001620
       END                                                                   001630
       SUBROUTINE MANIPUL(A,3,C,K),RETURNS(N2)                               001640
       COMMON/NEXT/ NOTHING                                                  001650
       DIMENSION KEY(7)                                                      001660
       DIMENSION A(15),                    LARRAY(29),       I(5,3,3),       001670
      *      C1(4),        C2(6),          MARRAY(23,2)                     001680
       LOGICAL L2                                                            001690
       L2=.TRUE.                                                             001700
       IF(C1(2).EQ.K)      L2=.FALSE.                                        001710
C      T(1,1,1)=25+3-7*K LARRAY(K)*MARRAY(K,1)                               001720
       ITTTTAA=IIADD(25.5,7HMANIPUL,  9,0,KEY)                               001730
       ITTTTA9=IIMPY(7.K,7HMANIPUL,  9,0,KEY)                                001740
       ITTTTAC=IIMPY(ITTTTA9,LARRAY(K),7HMANIPUL,  9,0,KEY)                  001750
       ITTTTA9=IIMPY(ITTTTAC,MARRAY(K,1),7HMANIPUL,  9,0,KEY)                001760
       T(1,1,1)=IIMNS(ITTTTAA,ITTTTA9,7HMANIPUL,  9,1,KEY)                   001770
 17   CONTINUE                                                              001780
C      IF((LARRAY(7)+.).EQ.LARRAY(7+.))     LARRAY(7)=0.1                    001790
       IF((LARRAY(7)+.).EQ.LARRAY(7+.))                                      001800
      *GO TO 7911                                                            001810
       GO TO 7912                                                           001820
 7911 CONTINUE                                                              001830
       LARRAY(7)=RASGN(0.1,7HMANIPUL, 10,0,KEY)                             001840
 7912 CONTINUE                                                              001850
       RETURN                                                               001860
       END                                                                  001870
       INTEGER FUNCTION IFUNC2(I1)                                          001880
       DIMENSION   KEY(7)                                                   001890
       CALL SETNBIT(24,1,1,1,17,6,1,KEY)                                    001900
C      IFUNC2=FUNC2(I1+ +I1,I1,5)                                           001910
       ITTTTA4=IIMPY(I1, +,7HIFUNC2 ,  +,6,KEY)                             001920
       ITTTTA3=IIADD(ITTTTAA,I1,7HIFUNC2 ,  +,0,KEY)                        001930
       ITTTTA5=IASGN(I1, HIFUNC2 ,  +,5,KEY)                                001940
       ITTTTA7=IASGN(5,7HIFUNC2 ,  +,3,KEY)                                 001950
       IFUNC2=RASGN(FUNC2(ITTTTA9,ITTTTAC,ITTTTA3),7HIFUNC2 ,  +,1,KEY)     001960
C      IF(IFUNC2.GT.5.0)     IFUNC2=0                                       001970
       IF(IFUNC2.GT.5.0)                                                    001980
       GO TO 7911                                                           001990
       GO TO 7912                                                           002000
 7911 CONTINUE                                                              002010
       IFUNC2=IASGN(0,7HIFUNC2 ,  +,1,KEY)                                  002020
 7912 CONTINUE                                                              002030
```

Figure 31 (Continued)

```
      RETURN
      END
      REAL FUNCTION FUNC3(I,I2,I3)
      DIMENSION KEY(7)
      CALL SETNBIT(12,1,3,1,7,5,1,KEY)
C     FUNC3=I/I2/I3*(I2+I3*(45.5+I2))
      ITTTTAA=IIDVD(I,I2,7HFUNC3  ,  4,0,KEY)
      ITTTTAB=IIDVD(ITTTTAA,I3,7HFUNC3  ,  4,0,KEY)
      RTTTTAA=R2ADD(45.5,I2,7HFUNC3  ,  4,0,KEY)
      RTTTTAB=R1MPY(I3,RTTTTAA,7HFUNC3  ,  4,0,KEY)
      RTTTTAC=R1ADD(I2,RTTTTAB,7HFUNC3  ,  4,0,KEY)
      FUNC3=R1MPY(ITTTTAB,RTTTTAC,7HFUNC3  ,  4,1,KEY)
      IF(FUNC3.LE.7D)   GO TO 40
C     FUNC3=FUNC3*FUNC3
      FUNC3=RRMPY(FUNC3,FUNC3,7HFUNC3  ,  6,1,KEY)
 40   RETURN
      END
      SUBROUTINE ERRORS(DUMMY,DUM1,DUM2)
      DIMENSION A(23),B(1,33)
      AA=B+C+
      RESULT=A=3
      AAA=NAMETOOLONG+3-
      BADNUM=123+5678
C     RESULT=FUNC7(R,E,P*3,9+K*TOTAL,F,G,H,SIN(X),I)
      RTTTTAA=RASGN(E,7HERRORS  ,  7,0,KEY)
      RTTTTAB=RASGN(R,7HERRORS  ,  7,0,KEY)
      RTTTTAC=R2MPY(P,3,7HERRORS  ,  7,0,KEY)
      RTTTTAD=R1MPY(K,TOTAL,7HERRORS  ,  7,0,KEY)
      RTTTTAE=R1ADD(9,RTTTTAD,7HERRORS  ,  7,0,KEY)
      RTTTTAF=RASGN(F,7HERRORS  ,  7,0,KEY)
      RTTTTAG=RASGN(G,7HERRORS  ,  7,0,KEY)
      RTTTTAH=RASGN(H,7HERRORS  ,  7,0,KEY)
      RTTTTAI=RASGN(X,7HERRORS  ,  7,0,KEY)
      RTTTTAJ=RASGN(SIN(RTTTTAI),7HERRORS  ,  7,0,KEY)
      RTTTTAK=RASGN(RTTTTAJ,7HERRORS  ,  7,0,KEY)
      ITTTTAA=IASGN(I,7HERRORS  ,  7,0,KEY)
      RESULT=RASGN(FUNC7(RTTTTAB,RTTTTAA,RTTTTAC,RTTTTAE,RTTTTAF,RTTTT
     AG,RTTTTAH,RTTTTAK,ITTTTAA),7HERRORS  ,  7,1,KEY)
C     GOOD=K+L/3.567
      ITTTTAA=IIDVD(L,34567,7HERRORS  ,  8,0,KEY)
      GOOD=IIADD(K,ITTTTAA,7HERRORS  ,  8,1,KEY)
C     MII=(AX+34*(4.6+AAA))
      RTTTTAA=RRADD(4.6,AAA,7HERRORS  ,  9,0,KEY)
      RTTTTAB=R1MPY(34,RTTTTAA,7HERRORS  ,  9,0,KEY)
      MII=RRADD(AX,RTTTTAB,7HERRORS  ,  9,1,KEY)
C     BADNUM=123.45678901234567 7
      BADNUM=IASGN(1234567890123+5677,7HERRORS  , 10,0,KEY)
C     A(2,3)=-5+3/8
      ITTTTAA=IIDVD(3,8,7HERRORS  , 11,0,KEY)
      A(2,3)=IIADD(-5,ITTTTAA,7HERRORS  , 11,1,KEY)
C     A(1+K+1+2+1+K-1+2+3-1+4-1)=LONG+ARRAY+ARGUMNT
      RTTTTAA=R1ADD(LONG,ARRAY,7HERRORS  , 12,0,KEY)
      A(1+K+1+2+1+K-1+2+3-1+4-1)=RRADD(RTTTTAA,ARGUMNT,7HERRORS  , 12,1
     ,KEY)
      RETURN
      END
```

Figure 31 (Continued)

the n-bit simulation function calls generated). For example, the arithmetic statement in line 420 of Figure 30 translates into the statements shown in lines 590 through 630 of Figure 31. In this example, statements that would not normally appear if n-bit simulation were not performed are designated with a pointer in Figure 30.

Certain extremes were introduced into the original program to demonstrate the versatility and range of statements that could be handled by the preprocessor. A few of these extreme cases shown in Figure 31 were continuation lines (line 830), function references with greater than six arguments (line 2270), long array argument lists (line 2560), long integer and real numbers (line 2490), and multiple IF statements within a given routine (lines 920-1490). The latter case was used to verify that unique labels were being generated from a series of modified IF statements.

N-bit Simulation Subroutines

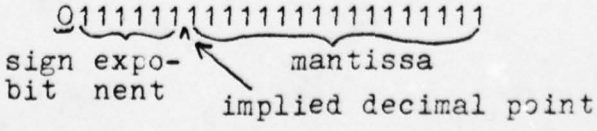
The preprocessor produced an n-bit simulated algorithm which referenced up to 22 different n-bit simulation function subroutines. Testing these n-bit simulation subroutines consisted of two phases. The first phase validated the key values which were computed by the SETNBIT subroutine. The second phase involved testing the 22 n-bit simulation function subroutines to verify the n-bit wordlength effects and other user options were being properly performed.

SETNBIT Key Values Verification

A critical portion of testing the n-bit simulation tool involved verifying the correctness of the key values computed by the SETNBIT subroutine. This is important because the performance of all 22 n-bit simulation subroutines are based upon the computed key values. The seven key values produced by the SETNBIT subroutine, based on the user options, are listed below:

1) Maximum positive fixed point value possible
2) Maximum positive floating point value possible
3) Least significant positive floating point value possible
4) Rounding or truncation effects flag
5) Overflow message printing or suppression flag
6) Number of least significant bits (LSB) to be truncated from floating point values for final assignments
7) Number of LSB to be truncated from floating point values for intermediate assignments

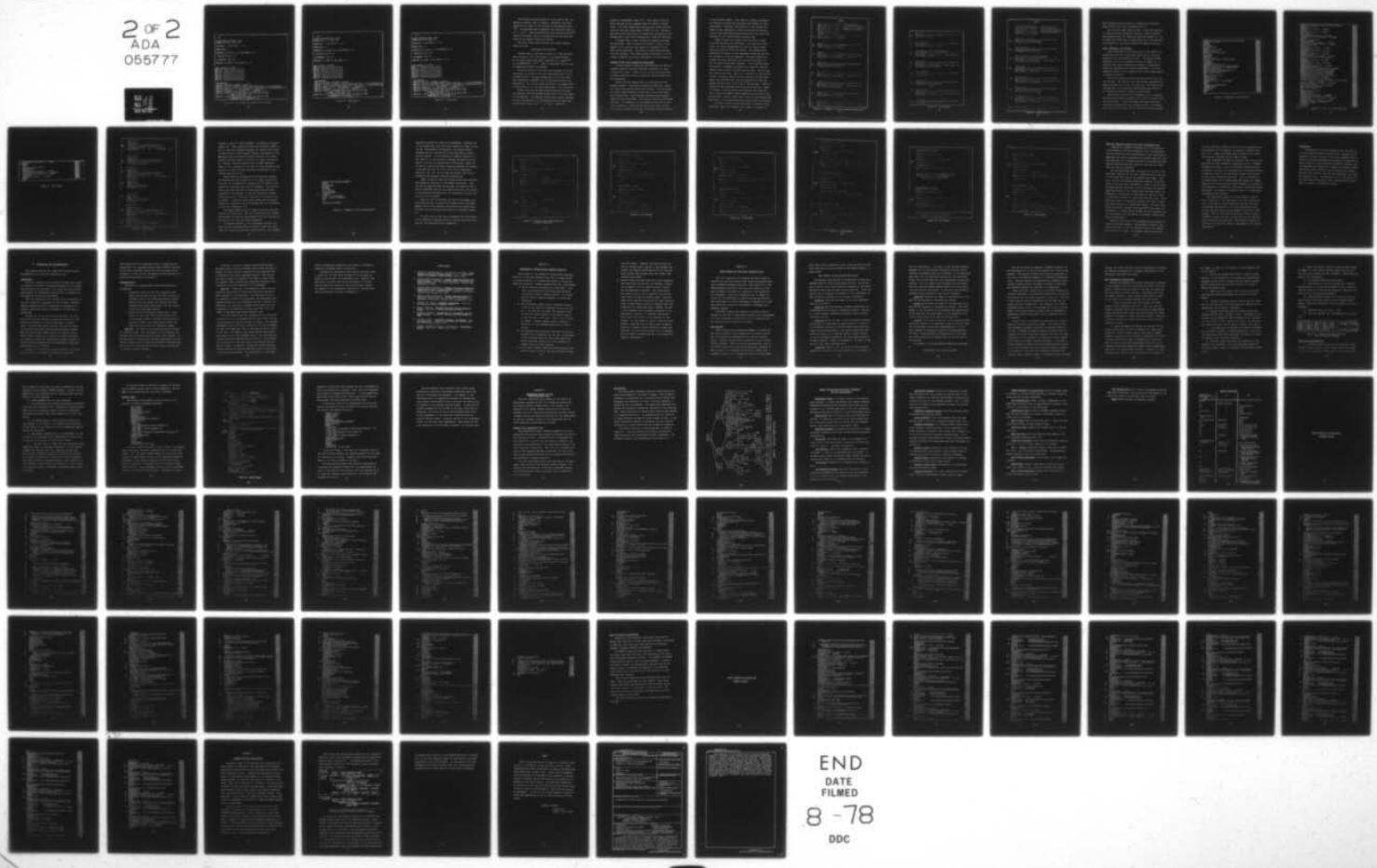
Figure 32 shows key values computed for various user options. Figure 32(a) has specifications for a 24-bit floating point word with 17 mantissa bits, six exponent bits, and an implied decimal point to the left of the mantissa. In a 24-bit word, the maximum floating point value representable would look like:



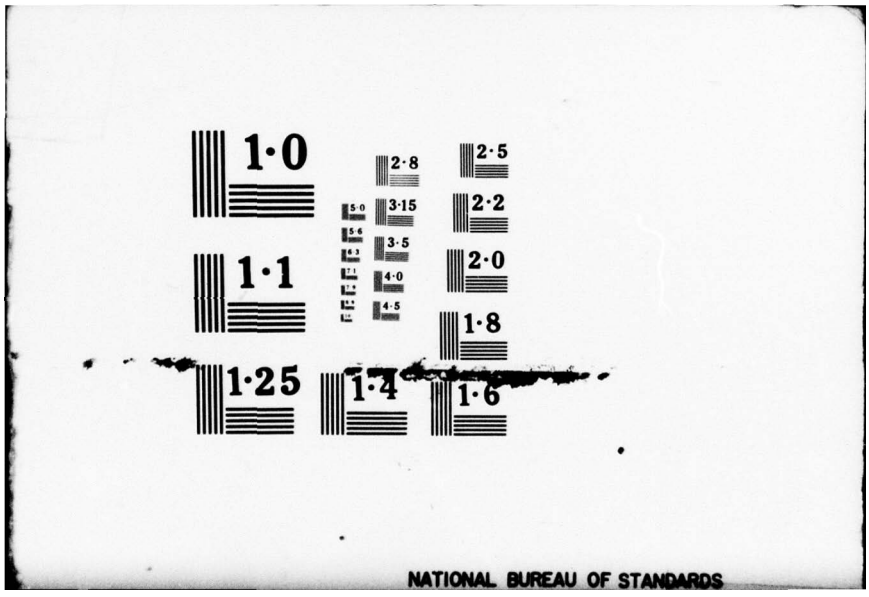
AD-A055 777 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
SOFTWARE TOOL(S) FOR EVALUATING THE EFFECTS OF FINITE WORDLENGT--ETC(U)
DEC 77 G A KLEIN

UNCLASSIFIED AFIT/GCS/EE/77-6 NL

2 OF 2
ADA
055777



END
DATE
FILMED
8-78
DDC


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
NATIONAL BUREAU OF STANDARDS

```
  (a)

N-BIT SPECIFICATIONS ARE:
NUMBER OF BITS/WORD =24

TRUNCATE (0) OR ROUND (1) =0

PRECISION = 1

OVERFLOW MESSAGES(1)/NO MESSAGES(0) =1

# MANTISSA BITS =17

# EXPONENT BITS =6

DECIMAL PT LEFT(0) OR RIGHT(1) = 0


 KEY VALUES IN OCTAL
KEY(1)= 000000000000377777777
KEY(2)= 175677777600000000000
KEY(3)= 166040000000000000000
KEY(4)= 000000000000000000000
KEY(5)= 000000000000000000001
KEY(6)= 000000000000000000037
KEY(7)= 000000000000000000037

 (IN DECIMAL)
 KEY(1)= MAXIMUM INTEGER = 8388607
 KEY(2)= MAXIMUM FLOATING POINT VALUE POSSIBLE = 2.147467264E+9
 KEY(3)= LEAST SIGNIFICANT FLOATING PT POSSIBLE = 2.328306436539E-10
 KEY(4)= ROUNDING (1)/ TRUNCATION (0) EFFECTS = 0
 KEY(5)= OVERFLOW MESSAGE PRINT(1)/SUPPRSSN(0) = 1
 KEY(6)= # BITS TRUNCATED FROM RIGHT OF FLTG PT FOR
              FINAL ASSIGNMENT = 31
 KEY(7)= # BITS TRUNCATED FROM RIGHT OF FLTG PT FOR
              INTERMEDIATE ASSIGNMENT = 31
```

Figure 32 Key Values For Various User Options

```
   (b)
N-BIT SPECIFICATIONS ARE:
NUMBER OF BITS/WORD =24

TRUNCATE (0) OR ROUND (1) =0

PRECISION = 1

OVERFLOW MESSAGES(1)/NO MESSAGES(0) =1

# MANTISSA BITS =17

# EXPONENT BITS =6

DECIMAL PT LEFT(0) OR RIGHT(1) = 1


 KEY VALUES IN OCTAL
KEY(1)= 000000000000037777777
KEY(2)= 200077777600000000000
KEY(3)= 170140000000000000000
KEY(4)= 000000000000000000000
KEY(5)= 000000000000000000001
KEY(6)= 000000000000000000037
KEY(7)= 000000000000000000037

 (IN DECIMAL)
 KEY(1)= MAXIMUM INTEGER = 8388607
 KEY(2)= MAXIMUM FLOATING POINT VALUE POSSIBLE = 2.81472829227E+14
 KEY(3)= LEAST SIGNIFICANT FLOATING PT POSSIBLE = .000030517578125
 KEY(4)= ROUNDING (1)/ TRUNCATION (0) EFFECTS = 0
 KEY(5)= OVERFLOW MESSAGE PRINT(1)/SUPPRSSN(0) = 1
 KEY(6)= # BITS TRUNCATED FROM RIGHT OF FLTG PT FOR
             FINAL ASSIGNMENT = 31
 KEY(7)= # BITS TRUNCATED FROM RIGHT OF FLTG PT FOR
             INTERMEDIATE ASSIGNMENT = 31
```

Figure 32 (Continued)

```
   (c) .
N-BIT SPECIFICATIONS ARE:
NUMBER OF BITS/WORD =24

TRUNCATE (0) OR ROUND (1) =1

PRECISION = 2

OVERFLOW MESSAGES(1)/NO MESSAGES(0) =1

# MANTISSA BITS =17

# EXPONENT BITS =6

DECIMAL PT LEFT(0) OR RIGHT(1) = 0


 KEY VALUES IN OCTAL
KEY(1)= 000000000000377777777
KEY(2)= 175677777777777777600
KEY(3)= 166040000000000000000
KEY(4)= 000000000000000000001
KEY(5)= 000000000000000000001
KEY(6)= 000000000000000000037
KEY(7)= 000000000000000000007

 (IN DECIMAL)
 KEY(1)= MAXIMUM INTEGER = 8388607
 KEY(2)= MAXIMUM FLOATING POINT VALUE POSSIBLE = 2.147483647999E+9
 KEY(3)= LEAST SIGNIFICANT FLOATING PT POSSIBLE = 2.328306436539E-10
 KEY(4)= ROUNDING (1)/ TRUNCATION (0) EFFECTS = 1
 KEY(5)= OVERFLOW MESSAGE PRINT(1)/SUPPRSSN(0) = 1
 KEY(6)= # BITS TRUNCATED FROM RIGHT OF FLTG PT FOR
              FINAL ASSIGNMENT = 31
 KEY(7)= # BITS TRUNCATED FROM RIGHT OF FLTG PT FOR
              INTERMEDIATE ASSIGNMENT = 7
```

Figure 32 (Continued)

The maximum exponent possible for six bits is ± 31, assuming an exponent bias of 100000_2. Therefore, the value shown would be equal to the contents of the mantissa times 2^{+31}. The mantissa above represents the fractional value of $(2^{17} - 1)/2^{17}$. When multiplied out, this fraction times 2^{+31} equals 2147467264., which is consistent with the value computed by the SETNBIT routine for KEY(2).

The least significant floating point value possible would look like

0000000,10000000000000000

The mantissa in this case is equal to .5 and the exponent is equal to -31 (with respect to the bias 100000_2). So the least significant number possible is .5 times 2^{-31} or $2.328306436539 * 10^{-10}$. This is consistent with the value computed for KEY(3) by SETNBIT.

The verification of the fixed point maximum can be accomplished by looking at the octal representation of KEY(1) in Figure 32(a). The least significant 23 bits are ones; this translates to 8388607 for its equivalent decimal representation. In a 24-bit fixed point word, the largest value would have 1's in the least significant 13 bit positions.

Figure 32(b) has the same specifications as (a) with the exception that the implied decimal point is to the right of the mantissa. The maximum floating point number possible for this representation is the maximum mantissa, which has a fixed point value of $2^{17} - 1$ or 131071, times 2^{+31},

yielding 2.81472829227 times 10^{14}. This value is also exactly the same as the computed value for KEY(2) of Figure 32(b). The least significant floating point value also computes to the exact value shown in KEY(3) of (b). The maximum fixed point value does not change with the moving of the decimal point since only floating point values are affected.

In Figure 32(c), double precision arithmetic accuracy was specified. KEY(7) contains the number 7, which is the number of bit positions that should be truncated off the CDC's 48-bit mantissa. This means 41 significant mantissa bits are retained. The single precision mantissa of 17 bits (shown in KEY(6)) plus extra wordlength of 24 bits equals 41.

Testing Of The N-bit Simulation Subroutines

A unique function subroutine was developed for each arithmetic operation and each possible combination of input variable data types. Twenty of the 22 function subroutines developed perform arithmetic operations and two perform simple assignments.

The 20 function subroutines can be divided into two general classes: those which produce floating point values and those which produce fixed point values. The subroutines of a given class basically differ only in the first statement they execute. That statement performs the arithmetic operation being simulated. After which, they are constructed the same. For instance, for the operations A*B and C+D the first statement in each handling subroutine would perform the required operation upon the operands and store the result

in the variable RESULT. From there on, exactly the same n-bit simulation effects are performed upon RESULT for both subroutines. Therefore, verification of the results produced by each combination of user options for one subroutine of each class should sufficiently prove the other subroutines of that class perform correctly also.

Figure 33 shows the results from two sets of fixed point and floating point additions. For each case within a set, the octal representation of the two values being added are shown by INPUT1 and INPUT2. The NORMAL OUTPUT represents the result of the computation using the full 60-bit precision of the CDC computer. The N-BIT OUTPUT represents the value resulting from the n-bit simulation of the two inputs being added together. If an overflow condition occurs, a message is printed (e.g. case 1C), and the maximum value possible replaces the old value. The key values precede each case. Case one is executed for the options: 16-bits, rounding, single precision accuracy, overflow messages, nine mantissa bits, and six exponent bits with the implied decimal point to the left of the mantissa. Cases 1A through 1H were performed with fixed point addition. Cases 1I through 1P were performed with floating point addition. Case 1O shows an example where underflow for the 16-bit word was detected. Case 1N shows that 9 bits of significance were maintained in the mantissa and also shows a case of round up.

In order to verify correct detection of overflow, many borderline cases were examined (e.g. cases 1A through 1D).

SET 1

```
      KEY(1)= 0000000000000077777      MAXIMUM FIXED POINT
      KEY(2)= 1756777000000000000      MAXIMUM FLOATING POINT
      KEY(3)= 1660400000000000000      LEAST SIGNIFICANT FLTG POINT
      KEY(4)= 0000000000000000001      ROUNDING(1) OR TRUNCATION(0)
      KEY(5)= 0000000000000000011     OVERFLOW MESSAGE OPTION
      KEY(6)= 0000000000000000147
      KEY(7)= 0000000000000000047
       INTEGER INTEGER ADDITION
       MAX-15 + 15
      INPUT1=0000000000000077760 INPUT2=0000000000000000017
1A    NORMAL OUTPUT=0000000000077777
       N-BIT OUTPUT=0000000000077777


       MAX-1 +0
      INPUT1=0000000000000077776 INPUT2=0000000000000000000
1B    NORMAL OUTPUT=0000000000077776
       N-BIT OUTPUT=0000000000077776


       MAX-1 + 2
      INPUT1=0000000000000077776 INPUT2=0000000000000000002
1C    NORMAL OUTPUT=0000000000000000000
      **** OVERFLOW IN TESTPRG  LINE #     1 FOR ADDITION   ***
       N-BIT OUTPUT=0000000000000077777


       IMIN+3 + -2
      INPUT1=7777777777777700003 INPUT2=7777777777777777775
1D    NORMAL OUTPUT=7777777777777700001
       N-BIT OUTPUT=7777777777777700001


       MAX + 3000
      INPUT1=0000000000000077777 INPUT2=0000000000000005670
1E    NORMAL OUTPUT=0000000000000103567
      **** OVERFLOW IN TESTPRG  LINE #     1 FOR ADDITION   ***
       N-BIT OUTPUT=0000000000000077777


       -MAX + -20
      INPUT1=7777777777777700000 INPUT2=7777777777777777753
1F    NORMAL OUTPUT=7777777777777677754
      **** OVERFLOW IN TESTPRG  LINE #     1 FOR ADDITION   ***
       N-BIT OUTPUT=7777777777777700000


       MAX + - MAX
      IN-UT1=0000000000000077777 INPUT2=7777777777777700000
1G    NORMAL OUTPUT=0000000000000000000
       N-BIT OUTPUT=0000000000000000000


       (MAX-1) + 35
      INPUT1=0000000000000077776 INPUT2=0000000000000000043
1H    NORMAL OUTPUT=0000000000000100041
      **** OVERFLOW IN TESTPRG  LINE #     1 FOR ADDITION   ***
       N-BIT OUTPUT=0000000000000077777
```

Figure 33 Examples Of N-bit Simulation Additions

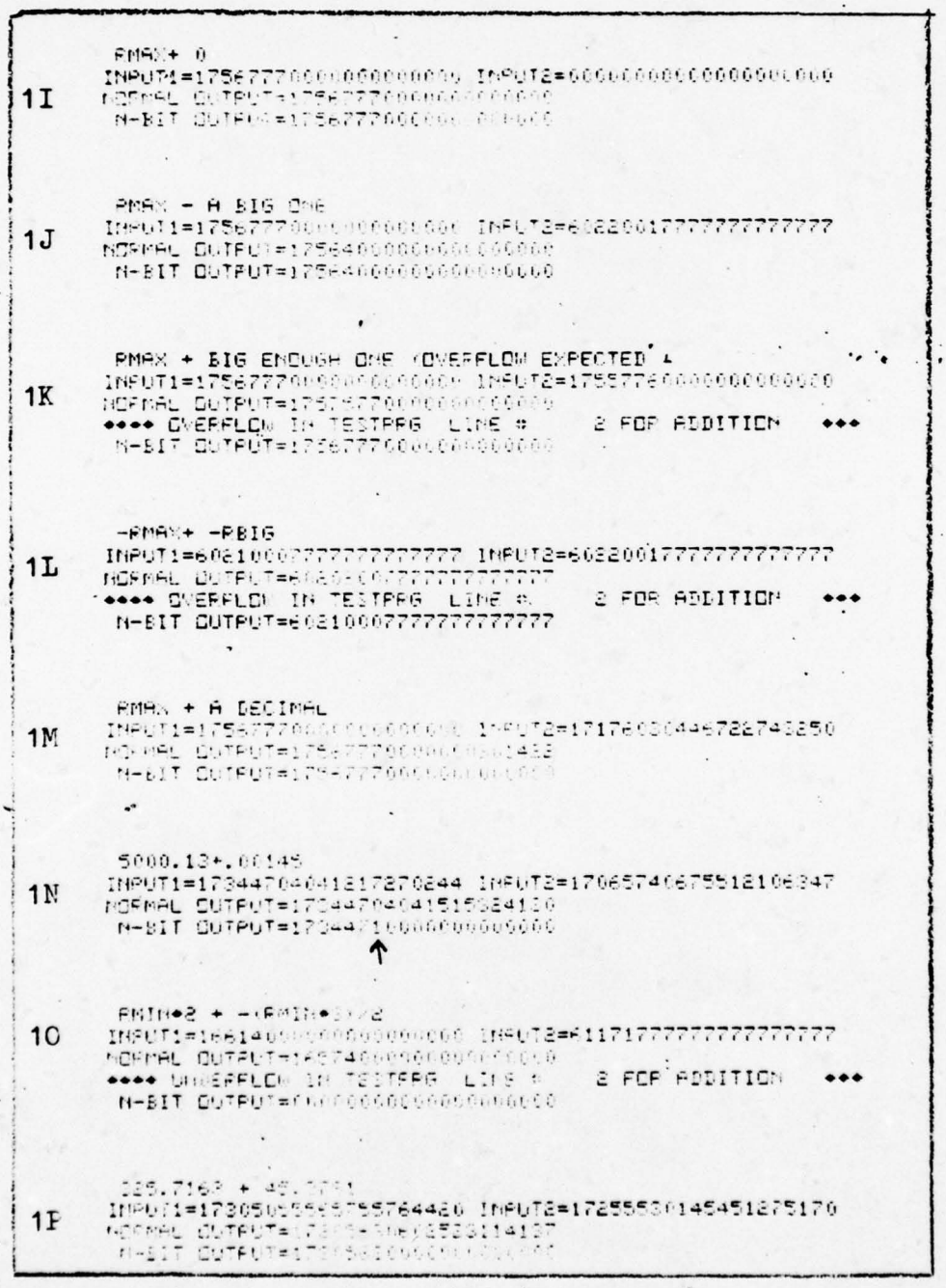


Figure 33 (Continued)

SET 2

```
KEY(1)= 0000000000000000077777     MAXIMUM FIXED POINT
KEY(2)= 1756777777777777777600     MAXIMUM FLOATING POINT
KEY(3)= 1660400000000000000000     LEAST SIGNIFICANT FLTG POINT
KEY(4)= 0000000000000000000001     ROUNDING(1) OR TRUNCATION(0)
KEY(5)= 0000000000000000000001    OVERFLOW MESSAGE OPTION
KEY(6)= 0000000000000000000047
KEY(7)= 0000000000000000000007
```

2A

```
 RMAX+ 0
INPUT1=1756777777777777777600 INPUT2=0000000000000000000000
NORMAL OUTPUT=1756777777777777777600
 N-BIT OUTPUT=1756777777777777777600
```

2B

```
 RMAX - A BIG ONE
INPUT1=1756777777777777777600 INPUT2=6022001777777777777777
NORMAL OUTPUT=1756400777777777777600
 N-BIT OUTPUT=1756400777777777777600
```

2C

```
 RMAX + BIG ENOUGH ONE (OVERFLOW EXPECTED
INPUT1=1756777777777777777600 INPUT2=1755776000000000000000
NORMAL OUTPUT=1757577377777777777700
**** OVERFLOW IN TESTPRG  LINE #      2 FOR ADDITION    ***
 N-BIT OUTPUT=1756777777777777777600
```

2D

```
 -RMAX+ -RBIG
INPUT1=6021000000000000000177 INPUT2=6022001777777777777777
NORMAL OUTPUT=6020200400000000000077
**** OVERFLOW IN TESTPRG  LINE #      2 FOR ADDITION    ***
 N-BIT OUTPUT=6021000000000000000177
```

2E

```
 RMAX + A DECIMAL
INPUT1=1756777777777777777600 INPUT2=17176030446722743250
NORMAL OUTPUT=1757400000000000141511
**** OVERFLOW IN TESTPRG  LINE #      2 FOR ADDITION    ***
 N-BIT OUTPUT=1756777777777777777600
```

2F

```
 5000.13+.00145
INPUT1=17344704041217270244 INPUT2=17065740675512106247
NORMAL OUTPUT=17344704041515224130
 N-BIT OUTPUT=17344704041515224200
                                ↑
```

2G

```
 RMIN*2 + -(RMIN*3) 2
INPUT1=1661400000000000000000 INPUT2=6117177777777777777777
NORMAL OUTPUT=1657400000000000000000
**** UNDERFLOW IN TESTPRG  LINE #      2 FOR ADDITION    ***
 N-BIT OUTPUT=0000000000000000000000
```

2H

```
 325.71E3 + 45.3791
INPUT1=[illegible] INPUT2=[illegible]
NORMAL OUTPUT=[illegible]
 N-BIT OUTPUT=[illegible]
```

Figure 33 (Continued)

Also computations which result in values well within the range of the 16-bit word are shown in Set 1.

Set 2 has the same options as Set 1, with the exception of triple precision arithmetic accuracy. Cases 2F and 2H show examples where the resulting value was rounded up. Also these cases show that 41 significant bits of the mantissa are retained for the double precision arithmetic accuracy.

N-bit Simulation Tool Effects

After verifying the two major parts of the system (i.e. preprocessor and n-bit simulation subroutines), the n-bit simulation tool was tested as a whole. The sample FORTRAN algorithm to be n-bit simulated is shown in Figure 34. It performs very simple computations designed to demonstrate losses of significance for shorter wordlengths. The n-bit simulated version of the program (built by the preprocessor) is shown in Figure 35.

The n-bit simulated program was then executed with various options. The results from these various executions are shown in Figure 36 (b), (c), (d), and (e). The results from the execution of the original FORTRAN algorithm without n-bit simulation effects is shown in Figure 36(a).

Figure 36(b) used a 24 bit wordlength with 17 mantissa bits (with the implied decimal point to its left) and 6 bits for the exponent. It was performed with single precision arithmetic accuracy and with rounding effects. The resulting output for TOTAL was 376784. while the original program

```
PROGRAM TEST4(INPUT,OUTPUT,TAPE1=OUTPUT,TAPE2=INPUT)                    000100
DIMENSION K(10),KEY(7)                                                  000110
B1=55210.                                                               000160
B2=7.327                                                                000170
DO 8 I=1,50                                                             000180
TOTAL=B1/B2+TOTAL                                                       000190
CONTINUE                                                                000200
PRINT *," TOTAL=",TOTAL                                                 000210
HALF=TOTAL-TOTAL/2.                                                     000220
PRINT *," HALF TOTAL=",HALF                                             000230
X1=.5791123                                                             000240
X2=.5790995                                                             000250
DIFF=X1-X2                                                              000260
WTDIFF=DIFF*1000.                                                       000270
PRINT *," DIFF=",DIFF," WTDIFF=",WTDIFF                                 000280
X3=.97265                                                               000290
X4=.9857134                                                             000300
X34DIFF=X3-X4                                                           000310
RATIO=WTDIFF/X34DIFF*100                                                000320
PRINT *," X34DIFF=",X34DIFF," RATIO =",RATIO                            000330
FUNC=FUNC1(23456.34,5*45.,1,0,RATIO)*1.25                               000340
PRINT *," FUNC=",FUNC                                                   000350
CALL SUB1(RATIO,X34DIFF-.453,DIFF,TOTAL)                                000360
PRINT *," SUB RATIO=",RATIO                                             000370
STOP " TEST4 COMPLETE"                                                  000390
END                                                                     000400
FUNCTION FUNC1(A,B,I,J,R)                                               000410
DIMENSION KEY(7)                                                        000420
R1=A**(-2)+B**(-2)                                                      000450
FUNC1=R1/3000.                                                          000460
IF(FUNC1.LT..00025)   FUNC1=.00025                                      000470
RETURN                                                                  000480
END                                                                     000490
SUBROUTINE SUB1(R1,R2,R3,R4)                                            000500
DIMENSION KEY(7)                                                        000510
R1=R1+R2+R3+R4                                                          000540
R1=R1**(-2)                                                             000550
RETURN                                                                  000560
END                                                                     000570
```

Figure 34 Example Of An Algorithm

```
      PROGRAM TEST4(INPUT,OUTPUT,TAPE1=OUTPUT,TAPE2=INPUT)                  000100
      DIMENSION K(10),KEY(7)                                                000110
      R1=55210.                                                             000120
      R1=RASGN(55210.,7HTEST4   ,  3,0,KEY)                                 000130
      R2=7.327                                                              000140
      R2=RASGN(7.327,7HTEST4   ,  4,0,KEY)                                  000150
      DO 8  I=1,50                                                          000160
      TOTAL=R1/R2+TOTAL                                                     000170
      RTTTTAA=RRDVD(R1,R2,7HTEST4   ,  6,0,KEY)                             000180
      TOTAL=RRADD(RTTTTAA,TOTAL,7HTEST4   ,  6,1,KEY)                       000190
    8 CONTINUE                                                              000200
      PRINT *," TOTAL=",TOTAL                                               000210
      HALF=TOTAL-TOTAL/2                                                    000220
      RTTTTAA=R2DVD(TOTAL,2,7HTEST4   ,  9,0,KEY)                           000230
      HALF=RRMNS(TOTAL,RTTTTAA,7HTEST4   ,  9,1,KEY)                        000240
      PRINT *," HALF=",HALF                                                 000250
      X1=.5791123                                                           000260
      X1=RASGN(.5791123,7HTEST4   , 11,0,KEY)                               000270
      X2=.5790995                                                           000280
      X2=RASGN(.5790995,7HTEST4   , 12,0,KEY)                               000290
      DIFF=X1-X2                                                            000300
      DIFF=RRMNS(X1,X2,7HTEST4   , 13,1,KEY)                                000310
      WTDIFF=DIFF*1000.                                                     000320
      WTDIFF=RRMPY(DIFF,1000.,7HTEST4   , 14,1,KEY)                         000330
      PRINT *," DIFF=",DIFF," WTDIFF=",WTDIFF                               000340
      X3=.97265                                                             000350
      X3=RASGN(.97265,7HTEST4   , 16,0,KEY)                                 000360
      X4=.9857134                                                           000370
      X4=RASGN(.9857134,7HTEST4   , 17,0,KEY)                               000380
      X34DIFF=X3-X4                                                         000390
      X34DIFF=RRMNS(X3,X4,7HTEST4   , 18,1,KEY)                             000400
      RATIO=WTDIFF/X34DIFF*100                                              000410
      RTTTTA1=RRDVD(WTDIFF,X34DIFF,7HTEST4   , 19,0,KEY)                    000420
      RATIO=R2MPY(RTTTTAA,100.,7HTEST4   , 19,1,KEY)                        000430
      PRINT *," X34DIFF=",X34DIFF," RATIO=",RATIO                           000440
      FUNC=FUNC1(23456.34,5*45.,1,J,RATIO)*1.25                             000450
      RTTTTAA=R1MPY(5.,45.,7HTEST4   , 21,0,KEY)                            000460
      RTTTTAB=RASGN(23456.34,7HTEST4   , 21,0,KEY)                          000470
      ITTTTAA=IASGN(1,7HTEST4   , 21,0,KEY)                                 000480
      ITTTTAB=IASGN(J,7HTEST4   , 21,0,KEY)                                 000490
      RTTTTAC=RASGN(RATIO,7HTEST4   , 21,0,KEY)                             000500
      RTTTTAD=RAS N(FUNC1(RTTTTAB,RTTTTAA,ITTTTAA,ITTTTAB,RTTTTAC)          000510
     *          ,7HTEST4   , 21,0,KEY)                                     000520
      FUNC=RRMPY(RTTTTAD,1.25,7HTEST4   , 21,1,KEY)                         000530
      PRINT *," FUNC=",FUNC                                                 000540
      CALL SUB1(RATIO,X34DIFF-.453,DIFF,TOTAL)                              000550
      PRINT *," SUB RATIO=",RATIO                                           000560
      STOP " TEST4 COMPLETE"                                                000570
      END                                                                   000580
      FUNCTION FUNC1(A,B,I,J,P)                                             000590
      DIMENSION KEY(7)                                                      000600
      R1=A**(-2)+B**(-2)                                                    000610
      RTTTTAA=R2EXP(A,-2,7HFUNC1   ,  3,0,KEY)                              000620
      RTTTTAB=R2EXP(B,-2,7HFUNC1   ,  3,0,KEY)                              000630
      R1=RRADD(RTTTTAA,RTTTTAB,7HFUNC1   ,  3,1,KEY)                        000640
      FUNC1=R1/3000.                                                        000650
      FUNC1=RRDVD(R1,3000.,7HFUNC1   ,  4,1,KEY)                            000660
      IF(FUNC1.LT..00025)  FUNC1=.00025                                     000670
      IF(FUNC1.LT..00025)                                                   000680
     *GO TO 7911                                                            000690
      GO TO 7912                                                            000700
7911  CONTINUE                                                              000710
```

Figure 35 N-bit Simulated Algorithm

```
      FUNC1=RASGN(.000025,7HFUNC1   ,   5,0,KEY)                    000720
  912 CONTINUE                                                      000730
      RETURN                                                        000740
      END                                                           000750
      SUBROUTINE SUB1(R1,R2,R3,R4)                                  000760
      DIMENSION KEY(7)                                              000770
      R1=R1+R2+R3+R4                                                000780
      RTTTTAA=RRADD(R1,R2,7HSUB1    ,  3,0,KEY)                     000790
      RTTTTAB=RRADD(RTTTTAA,R3,7HSUB1    ,  3,0,KEY)                000800
      R1=RRADD(RTTTTAB,R4,7HSUB1    ,  3,1,KEY)                     000810
      R1=R1**(-2)                                                   000820
      R1=R2EXP(R1,-2,7HSUB1    ,  4,1,KEY)                          000830
      RETURN                                                        000840
      END
```

Figure 35 (Continued)

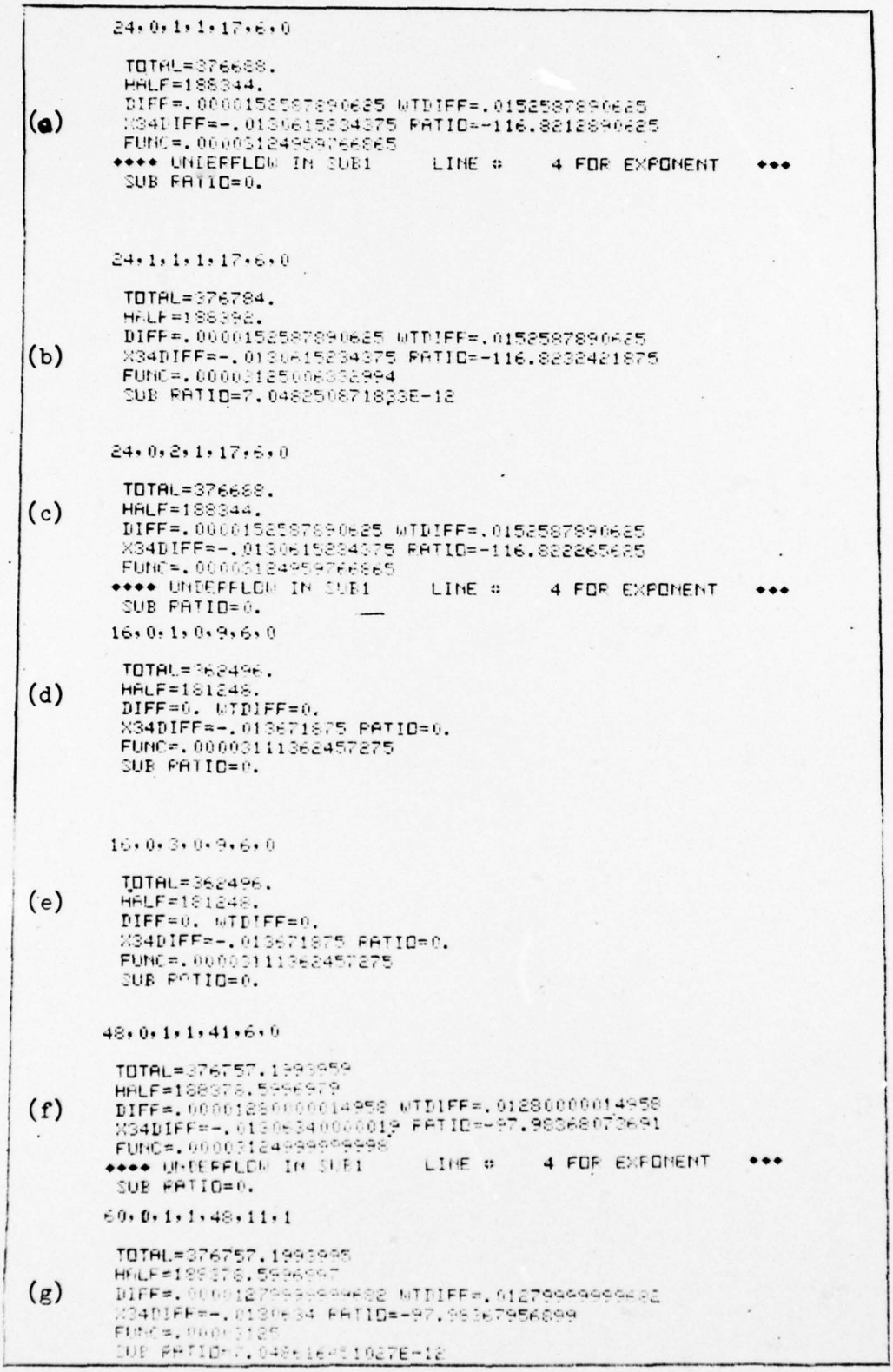


Figure 26 Sample Outputs

obtained a result of 376757.1993995, a difference of approximately +24. When truncation effects were applied (shown in (c)) to the n-bit simulated program, the results were slightly less accurate, TOTAL equalled 376688, a difference of -69. Employing double precision arithmetic accuracy with single precision storage (shown in (d)) did not impact the accuracy. However, double precision 24-bit storage variables, shown in (f), increased the accuracy of this algorithm's computation to almost the exact accuracy accomplished by the original algorithm in (a).

Double precision storage can be simulated by keeping the exponent the same length as in the single precision 24-bit word, but increasing the mantissa and the overall n-bit length by 24 (another full word of accuracy). The only caution a user must exercise in simulating double precision variables in this manner, is that the fixed point maximum values computed for detecting fixed point overflow will not be correct. Therefore, fixed point values that would have resulted in overflow in an n-bit machine may not be detected by the n-bit simulation tool.

The results shown in (g) of Figure 36 are for the full 60-bits of the CDC word. The n-bit simulated 60-bit results were exactly the same as those produced from the original algorithm and the CDC's full 60-bit accuracy.

For another example of n-bit wordlength effects, the small FORTRAN algorithm shown in Figure 37 was n-bit simulated for various combinations of user options. The original

```
      PROGRAM TEST1(INPUT,OUTPUT)
      SUM=0.
      A=.0001579
      B=2.123
      C=1.0319
      DO 5 I=1,100
      SUM=A+B*C+SUM
5     CONTINUE
      PRINT *," SUM=",SUM
      STOP " TEST1 COMPLETE"
      END

SUM=219.6930889999
```

Figure 37 Example Of A Second Algorithm

algorithm resulted in a SUM of 219.693088999. Numerous other bit lengths were tried and their results are shown in Figure 38. Particularly interesting is the varying effect rounding has upon the results of this algorithm at different bit lengths. A 12-bit word with rounding resulted in a SUM equal to 11, while without rounding SUM equalled 128 as shown in (a) and (d) respectively of Figure 38. Figure 38 (b) and (e) show a case where rounding increased the accuracy of the SUM from 204.5 to 213.5 for a 16-bit wordlength. Cases (f), (g), (h), and (i) show the results from various wordlength and exponent, mantissa combinations.

Cases (j) and (k) show user options which were incorrect and flagged as erroneous by the SETNBIT subroutine. Case (j) had too many mantissa and exponent bits specified for a 16-bit wordlength and case (k) specified 64-bits, but 60-bits is the maximum wordlength this n-bit simulation tool can simulate on the CDC machine.

Cases (l) and (m) contrast the results obtainable from this algorithm with a single word storage 24-bit wordlength machine and a 24-bit machine with double word storage capabilities, both involving single precision arithmetic accuracy.

It would be up to the user to determine the best choice of bit length and characteristics to provide sufficient accuracy for the algorithms being considered.

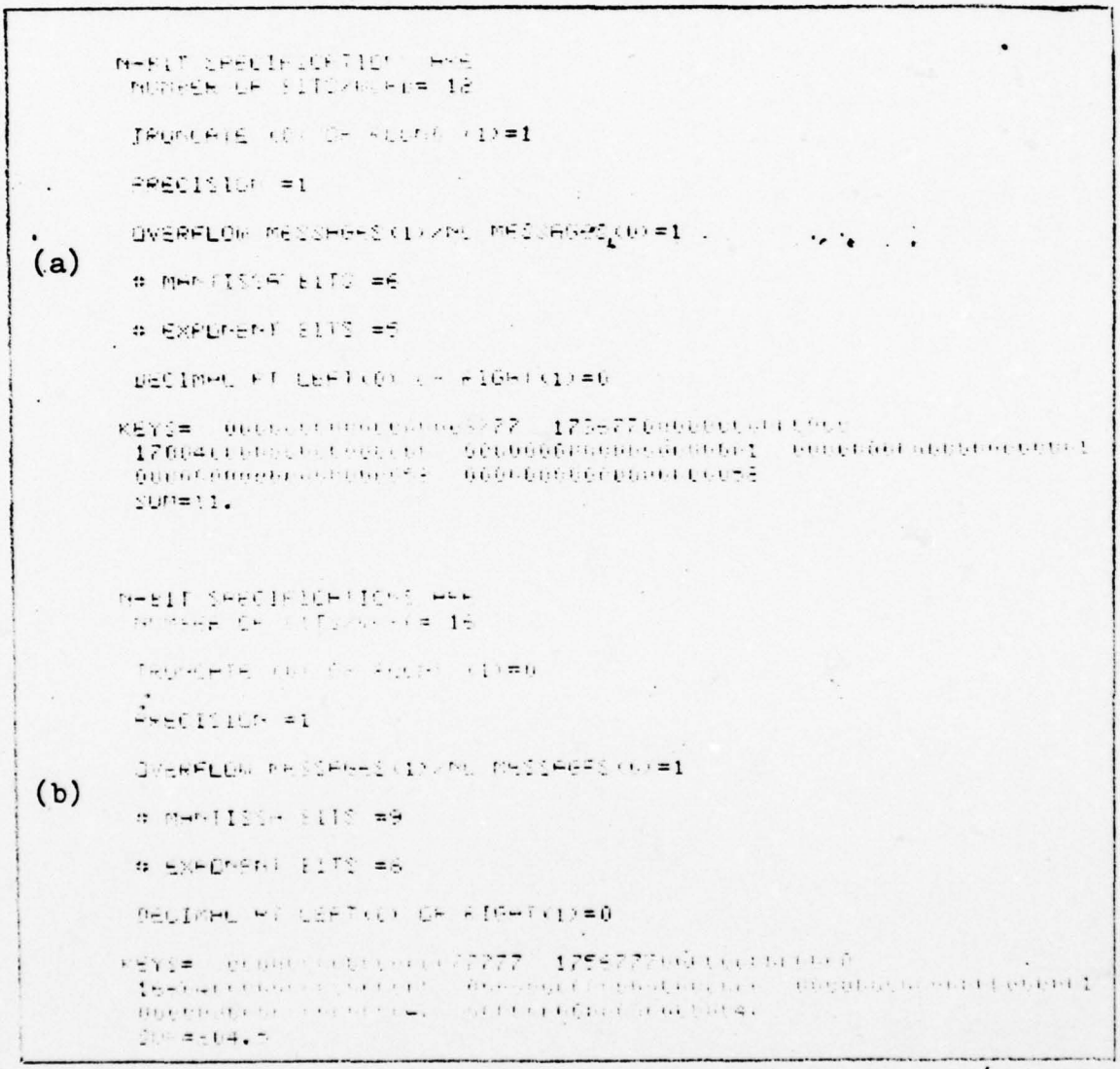


Figure 38 Results From N-bit Simulation
Of Second Algorithm

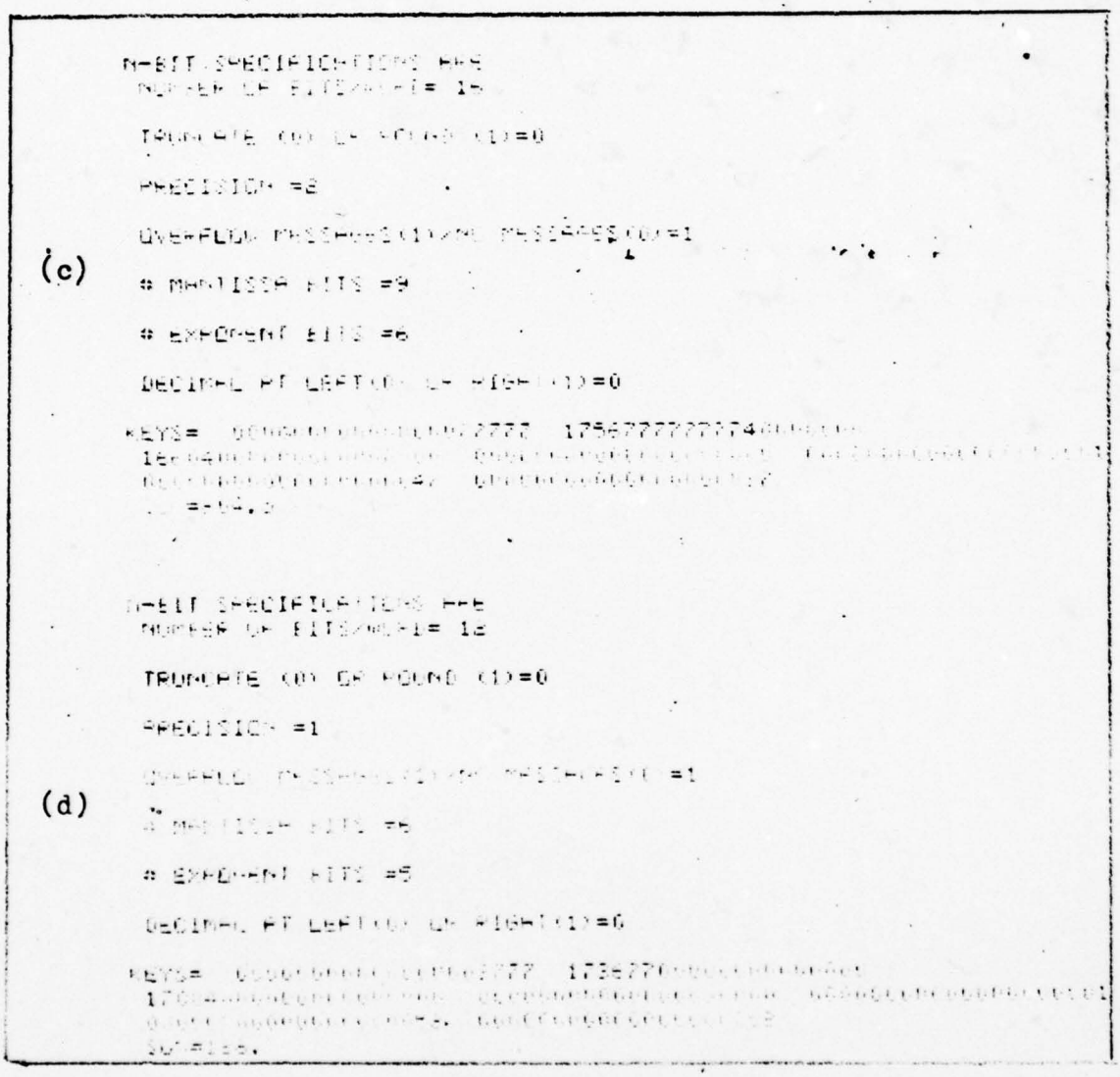


Figure 38 (Continued)

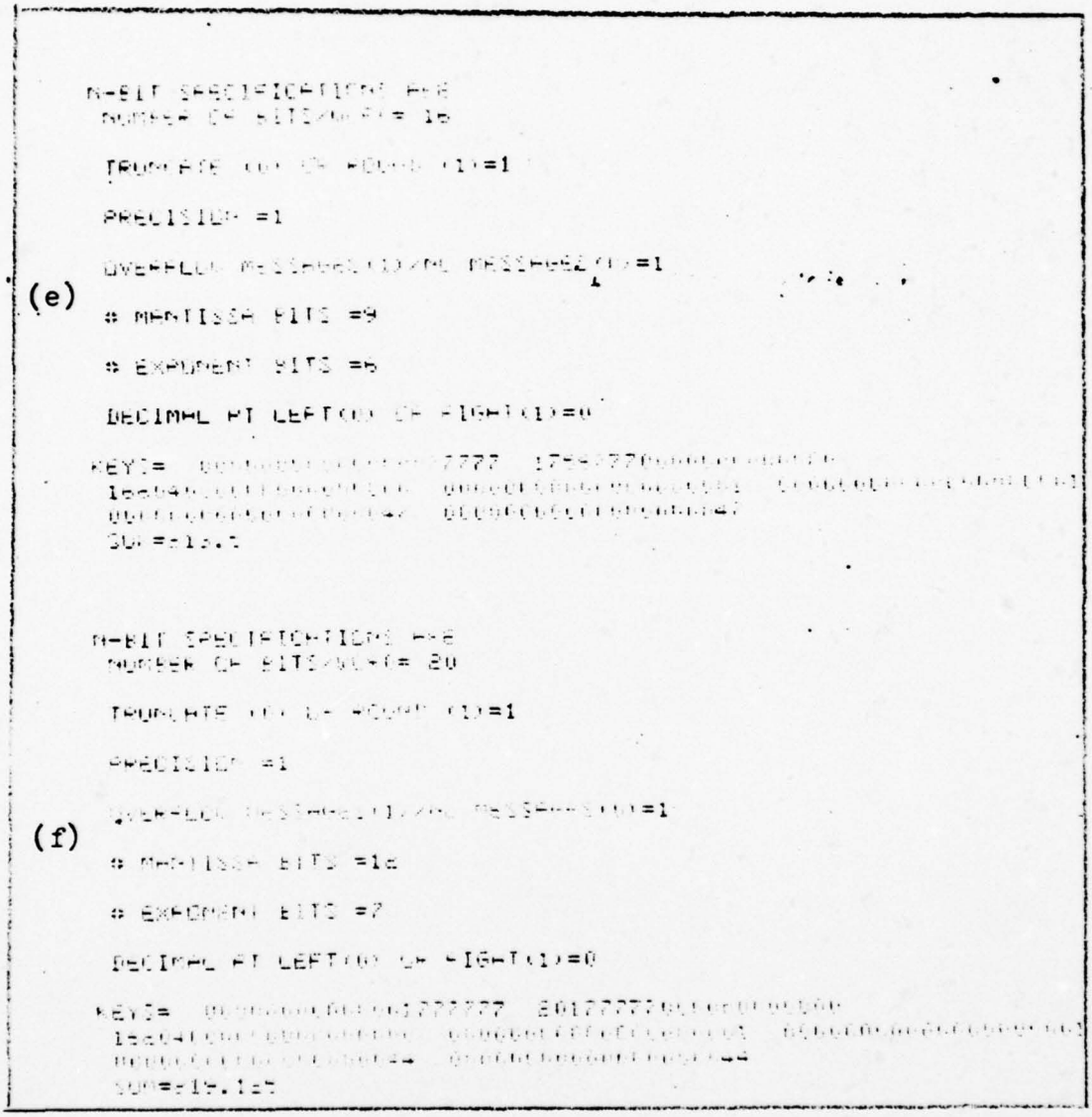
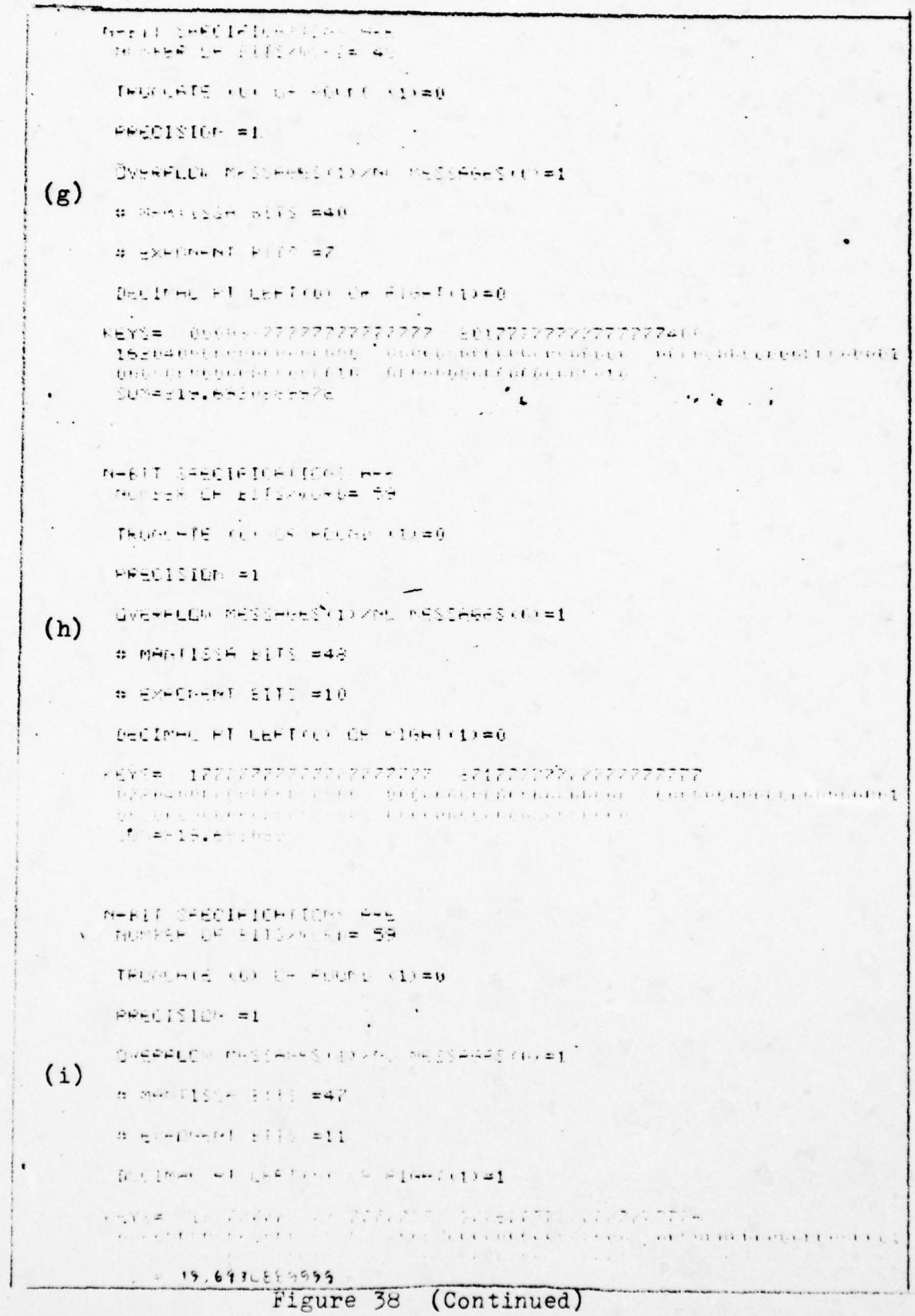


Figure 38 (Continued)



(g)

(h)

(i)

```
      N-BIT SPECIFICATIONS ARE:
      NUMBER OF BITS/WORD =16

      TRUNCATE (0) OR ROUND (1) =1

      PRECISION = 1

      OVERFLOW MESSAGES(1)/NO MESSAGES(0) =1
(j)
      # MANTISSA BITS =10

      # EXPONENT BITS =7

      DECIMAL PT LEFT(0) OR RIGHT(1) = 0


      MANTISSA + EXPONENT + 1 SHOULD EQUAL 'N'-BITS
      ERROR- ILLEGAL SETBIT DATA PARAMETER FOUND
          STOP   BAD INPUT TO SETBIT ROUTINE
            .280 CP SECONDS EXECUTION TIME



      N-BIT SPECIFICATIONS ARE:
      NUMBER OF BITS/WORD =64

      TRUNCATE (0) OR ROUND (1) =0

      PRECISION = 1

      OVERFLOW MESSAGES(1)/NO MESSAGES(0) =1
(k)
      # MANTISSA BITS =53

      # EXPONENT BITS =10

      DECIMAL PT LEFT(0) OR RIGHT(1) = 1


      TOO MANY BITS SPECIFIED: N-BIT MAX IS 60  --- 64 WERE SPECIFIED
      ERROR- ILLEGAL SETBIT DATA PARAMETER FOUND
          STOP   BAD INPUT TO SETBIT ROUTINE
            .031 CP SECONDS EXECUTION TIME
```

Figure 38 (Continued)

(l)

```
N-BIT SPECIFICATIONS ***
 NUMBER OF BITS/WORD= 24

 TRUNCATE (0) OR ROUND (1)=1

 PRECISION =1

 OVERFLOW MESSAGES(1)/NO MESSAGES(0)=17

 # MANTISSA BITS =17

 # EXPONENT BITS =6

 DECIMAL PT LEFT(0) OR RIGHT(1)=0

KEYS=  00[illegible]777777  1757777760[illegible]
 1560[illegible]
 [illegible]
 SUM=19.714[illegible]
```

(m)

```
N-BIT SPECIFICATIONS ***
 NUMBER OF BITS/WORD= 48

 TRUNCATE (0) OR ROUND (1)=0

 PRECISION =1

 OVERFLOW MESSAGES(1)/NO MESSAGES(0)=1

 # MANTISSA BITS =41

 # EXPONENT BITS =6

 DECIMAL PT LEFT(0) OR RIGHT(1)=0

KEYS=  000[illegible]7777777777777777  1757777777777777760[illegible]
 15[illegible]
 [illegible]
 SUM =1[illegible].69366699974
```

Figure 38 (Continued)

Core And Execution Costs Of The N-bit Simulation Tool

Although not regarded originally as a limiting factor in the design of the n-bit simulation tool, the additional core and execution time requirements for n-bit simulating an algorithm may inhibit some users from readily using the tool. The resulting n-bit simulated version of a FORTRAN algorithm requires substantially more execution time and core space than the original algorithm.

For the algorithms shown in Figures 30, 34, and 37 the core space required for the n-bit simulated version was four to five times that used by the original algorithms. Moreover, the n-bit simulation subroutines added an extra 2000 words to the core size requirements. However, for the types of algorithms for which this tool is intended, the original algorithms usually do not require more than 5000 words. An expansion of these kinds of algorithms by a factor of four or five would probably not impact the user very much. Those algorithms that did exceed 5000 probably have much of their core space reserved for dimensioned arrays. The n-bit simulation tool would increase these algorithm's core space requirements to a lesser degree because the core expansion factor is dependent upon the number of arithmetic computations within an algorithm.

Execution time for an n-bit simulated algorithm may be a more significant problem. The programs in Figure 34 and 35 were modified so that the statements within the program were executed 500 times. The original algorithm required

.277 CPU seconds for execution while the n-bit simulated version required 2.4 CPU seconds, an increase of almost a factor of ten. Other tests found cases in which the CPU execution time was increased by more than 30 times.

This tremendous increase in execution time is due to the fact that for each arithmetic computation within a FORTRAN arithmetic statement, six to ten FORTRAN instructions must be executed to perform the necessary n-bit wordlength effects. In addition, there is the subroutine linkage time required to call n-bit simulation subroutine. This introduction of new subroutines into a FORTRAN algorithm causes the FORTRAN compiler to save all register values somewhere in storage before jumping to a subroutine, then restore those values after returning. An increase in execution time is not surprising considering these factors.

This additional execution time may greatly impact a user's readiness to take advantage of the n-bit simulation tool if the user's original algorithm required more than 120 seconds for execution. The n-bit simulation version of that algorithm could require anywhere from 1200 to 3600 seconds. Now the user would be dealing in terms of hours instead of minutes and that is only CPU time, not total run time, which would be even more. One method to lessen the execution time explosion factor is discussed in the following chapter.

Conclusions

In conclusion, the n-bit simulation tool was found to be an accurate and fairly versatile tool. Through the employment of various options, a user could simulate the effects of various wordlengths with rounding or truncation effects, double or triple precision arithmetic accuracy, various mantissa and exponent bit lengths, and simulate double wordlength storage variables (for floating point values). The tool does, however, cost the user additional core space and execution time. Nevertheless, it is a very useful tool for those applications where these limiting factors do not come into play to a substantial degree.

VI. Conclusions And Recommendations

This chapter provides the reader with conclusions and recommendations for the n-bit simulation tool.

Conclusions

The n-bit simulation tool provides the user with a means to evaluate the effects of varying wordlength upon the numerical accuracy of an algorithm. The results can then be evaluated against the performance specification to determine the wordlength and precision requirements.

The n-bit simulation tool requires that an algorithm being n-bit simulated is executed on the CDC 6600 or CYBER 74 computer systems and that the algorithm is expressed in a slightly restricted version of FORTRAN IV, as described in Appendix B.

Through the various user options available, the n-bit simulation tool can simulate the numerical effects for computations performed on a fairly wide range of computer types. By exercising various options, the user can specify a computer type which has various wordlengths, varying exponent and mantissa lengths (where the mantissa represents either a fractional or fixed point value), rounding or truncation effects for computations, single, double, or triple wordlength accumulators, and doubleword storage for wordlengths of up to 24-bits.

In addition, the user may specify different n-bit wordlength effects for different portions of an algorithm.

This feature would be especially useful in simulating the algorithms of an integrated system; where different portions of the total integrated system (such as the Kalman Filter, navigation, flight control, and weapon delivery systems) require differing amounts of precision.

Recommendations

Two areas for immediate/near future development are recommended:

1. Include an option for the n-bit simulation tool which would provide the user the exact results obtainable from an n-bit wordlength machine; specifically giving the values that overflow the exact representation which would result from an overflow of the n-bit wordlength machine.
2. Modify the n-bit simulation preprocessor to substitute in-line code, which would perform the n-bit wordlength effects, instead of substitution of subroutine calls. This change would improve the execution time of the n-bit simulated program.

Case One. The result of an overflow varies for different computer types. With the present n-bit simulation tool, maximum values are substituted for values that overflow. In the real world, it may be that the most significant bits are simply truncated off for fixed point addition overflows while multiplication overflows result in the correct mantissa and the maximum exponent possible.

An option to the n-bit simulator which would produce the exact results from an overflow, would enable the user to analyze the exact results obtainable from an algorithm that is processed on an n-bit wordlength machine. This new option would require an analysis of the ways in which different computer types overflow and incorporate simulation techniques into the n-bit simulation tool accordingly.

Case Two. Presently, n-bit simulation effects are performed by substitution of subroutine calls for each arithmetic operation of an algorithm. Upon execution of the n-bit simulation version of an algorithm, the subroutines referenced perform the n-bit wordlength effects. However, the additional time required to simulate these effects for all operations in an algorithm is a very high ratio (15 to 40 times) to the algorithms normal execution time.

It is anticipated that the introduction of in-line code to perform the n-bit wordlength effects would reduce the additional execution time by at least one-third. This proposed enhancement would allow the tailoring of the code, performed for the n-bit simulation task, to the user options in effect at the time. For example, the present system tests within each subroutine for the rounding option. The subroutine executes differently depending upon the option. By tailoring the in-line code to the user option, decision-making time would be saved. In addition, subroutine linkage time (saving and restoring of COMPASS registers) would be saved with the in-line code approach. The total amount of time saved

could be substantial depending on the number of arithmetic operations performed within an algorithm.

Although this enhancement would improve execution time, it would cost the user twice the amount of core space required by the present n-bit simulation tool. However, this additional core space may be more acceptable to the user than the additional execution time of the present system. The changes to the preprocessor required for this modification are described in detail in Appendix D.

Bibliography

1. Abd-alla, Abd-elfattah M., and Arnold C. Meltzer. Principles of Digital Computer Design. Vol.I. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1976.

2. Control Data Corporation. Control Data 6000 Series Computer Systems. Hardware Reference Manual (Revision AH. Pub. No. 60100000). St. Paul, Minnesota: Control Data Corporation, 1976.

3. Control Data Corporation. Control Data 6000 Series Computer Systems 7600 Computer System. Reference Manual (Revision D. Pub. No. 60279900). Sunnyvale, California: Control Data Corporation, 1972.

4. Control Data Corporation. FORTRAN Extended Version 4. Reference Manual (Revision J., Pub. No. 60305601). Sunnyvale, California: Control Data Corporation, 1975.

5. Katzan, Jr., Harry. Advanced Programming. New York: Van Nostrand Reinhold Company, 1970.

6. Myers, Glenford. Reliable Software Through Composite Design. New York: Petrocelli/Charter, 1975.

7. Ralston, Anthony. Introduction to Programming And Computer Science. New York: McGraw-Hill Book Company, 1971.

8. Struble, George. Assembler Language Programming: The IBM System/360. Reading, Massachusetts: Addison-Wesley Publishing Company, 1969.

9. Yourdon, Edward and Larry L. Constantine. Structured Design. New York: Yourdon Inc., 1975.

Appendix A

Approaches To Finding Extra COMPASS Registers

The solution to the problem of finding extra registers would be to find some standard value that is always stored in a known register or to devise some method of saving and restoring registers. Several possibilities were investigated:

1) The A0 register contents may be predictable. However, no consistencies were found and destroying its contents caused irregular results. A0 is used as the relative starting address in a block copy operation.
2) The CDC manuals stated it was a standard practice of CDC to use the B1/B7 registers to store the value 1. If this were the case, that register could be used as a spare register at anytime as long as it was restored to the value 1 when it was no longer needed. However, several cases of exception were found to this standard practice-so it was of no use.
3) The Exchange Jump instruction was known to save the contents of all registers, but its use is restricted to the operating system for switching between two central programs, leaving the first program in a useable state for later re-entry.
4) If CDC had a standard register saving/restoring convention similar to IBM's, the extra register problem

would be solved. However, CDC does not have one per se; instead before jumping to any external subroutine the register values which will be required later (within that program unit) are stored, then restored when needed.

5) Another theory was that since the A and X registers were associated one-for-one, the address in register A_n (where n is 1-5) could be used to recover the corresponding X register. This would enable the current value of the X register to be written over, since it could be restored later. Once one spare 60-bit register was found, all other registers could be saved using a series of register moves and write-outs to a known save area. However, the value of X register does not always correspond to the associated A register. A technique was devised which saved the contents of an X whenever it was changed. In that manner it could be restored later. However, there was still a need to have an associated pair of A6/X6 or A7/X6 registers to make this possible. All this would require a lot of analysis of the COMPASS code which was felt to be beyond the scope of this thesis.

Appendix B

User's Manual For The N-bit Simulation Tool

The n-bit simulation tool enables the user a means to evaluate the numerical effects that varying wordlengths have upon a FORTRAN-programmed algorithm. When applied to a FORTRAN-programmed algortihm, this tool will produce the near exact arithmetic accuracy as a computer of the specified n-bit wordlength. This n-bit simulation tool is useable on either the CDC 6600/CYBER 74 computer systems and requires that algorithms to be n-bit simulated be expressed in a somewhat restricted version of the FORTRAN IV (EXTENDED) programming language.

This User's Manual will describe the seven options available to the user in specifying different n-bit wordlengths, the using programmer's restrictions will be described, and the required control cards will be shown.

User Options

Seven options were considered necessary to provide the user with a versatile n-bit simulation tool which could be used to simulate the characteristics of various computer types. Different options could be applied to every routine within a program. These options are expressed by a call to the SETNBIT subroutine as the first executable statement of each routine (i.e. following the DIMENSION, COMMON, DATA statements, and so on). The general form of this subroutine

call, where each # represents a user option and KEY is a variable name which must be included as the eighth argument, is shown below:

```
CALL SETNBIT (#1,#2,#3,#4,#5,#6,#7,KEY)
```

The variable name KEY must be dimensioned to seven (e.g. DIMENSION KEY(7)) within each program routine. The KEY array is used to store values which are computed by the SETNBIT subroutine based upon the user options specified. These values are used to simulate the effects of n-bit wordlength for each arithmetic operation of an arithmetic statement.

Option #1 specifies the number of bits per wordlength (from 8 to 60 bits).

Option #2 specifies whether arithmetic operations will be performed with rounding or truncation effects. The value 1 would indicate rounding effects and 0 would indicate truncation effects.

Option #3 allows the user to specify whether arithmetic calculations are performed in single, double, or triple n-bit wordlength precision, with the results being stored as single precision quantities. This option essentially simulates the effects of a computer having a single, double, or triple wordlength accumulator but only single precision storage variables. Single is indicated by the value 1, double by 2, and triple precision by 3.

Option #4 gives the user the option to have overflow messages printed when they are detected (i.e. overflows of

the n-bit wordlength). The value 1 would indicate overflow messages will be printed when overflows occur and 0 would indicate the messages would not be printed. An overflow message would tell the user which routine and the line number within that routine where the n-bit wordlength overflow was detected. The line number would reference the line number from the original FORTRAN program (before the n-bit simulation modifications were made). In addition, the arithmetic operation being performed at the time is indicated.

Option #5 indicates the number of bits which are to be used for the mantissa of floating point values.

Option #6 indicates the number of bits which are to be used for the exponent of floating point values. The total number of bits specified for the mantissa and exponent must be one less than the total number of bits specified in option #1 (this other bit is used as the sign bit).

Option #7 specifies the position of the implied decimal point with respect to the mantissa, either to its left (making the mantissa a fractional representation) or to its right (making the mantissa a fixed point representation). The value 1 would position the decimal point on the right and 0 would position the decimal point on the left end of the mantissa.

An example of a typical SETNBIT subroutine call would be

```
CALL SETNBIT (16,1,2,0,9,6,0,KEY)
```

The options shown as arguments of SETNBIT specify a 16-bit wordlength with 9 bits in the mantissa and 6 bits in the exponent. The implied decimal point would be positioned to the left of the mantissa (making it a fractional representation). Rounding effects are specified for arithmetic calculations, double precision arithmetic accuracy is specified, and overflow messages are supposed to be suppressed.

Double precision storage can be simulated by adding the original wordlength to the mantissa specification (Option #5) and doubling the bit length specified in option #1. All other user options should be maintained as they were for the single precision storage word. For instance, for a 16-bit wordlength with 9 bits in the mantissa and 6 bits in the exponent, the double precision storage could be simulated by specifying a 32-bit wordlength, a 25 bit mantissa, and a 6 bit exponent.

Double precision arithmetic accuracy should be distinguished from double precision storage. With double precision arithmetic accuracy (which can be specified under option #3) a FORTRAN expressed arithmetic statement will be computed with double precision accuracy, but after the final computation of the statement is performed the result stored will contain only single precision accuracy. In contrast, double precision storage would have double precision accuracy, in the computation as well as double precision storage of the result. Since the n-bit simulation tool does not presently have a separate option to designate double precision

storage, the double precision can be accomplished by doubling the original wordlength and increasing the mantissa by one wordlength (discussed previously).

User Programming Restrictions

The user programming restrictions which must be adhered to for programs to be n-bit simulated are listed in this appendix. Restrictions are listed in the order in which they are most likely to impact the user. Each restriction will be briefly justified. Other things a user should be aware of are listed in a section following the restrictions.

1. Every routine must have the variable KEY dimensioned to seven (DIMENSION KEY (7)). Also, the first executable statement within each routine must be CALL SETNBIT (seven options and KEY), where the user specifies the seven options to be applied to that particular routine. These additions are required so each routine can perform the n-bit simulation effects.

2. Single assignment statements are required for all values coming from a source of different bit length specifications. Thus all values read in from cards or values passed as parameters from a routine of different n-bit specifications must be set equal to themselves, in order for them to take on their n-bit significance (since all n-bit simulated subroutines assume the values being operated upon have their n-bit significance already). Therefore, if the variable A was read into an n-bit routine then A should be

set equal to itself (A = A) before it is used anywhere else in the program.

3. The entire program must be free of syntax errors, before it is run through the n-bit simulation preprocessor. Debugging errors would be easier with the original program and the preprocessor has not been verified to be able to handle all syntax errors.

4. TAPE1 must be an output unit for each n-bit simulated program (overflow messages are printed to the unit number).

5. CONTINUE statements must be used as the last statement for DO Loops due to the way labels are handled by the preprocessor for arithmetic statements.

6. Variable data types must use default data types (Reals (A-H) and (O-Z), Integers (I-N)). Declaration of data types using INTEGER and REAL statements are not allowed, nor are IMPLICIT INTEGER statements allowed. The preprocessor selects handling routines based on default data types.

7. All arrays must be declared by DIMENSION statements (not by COMMON, INTEGER, or REAL declarations). This is so the preprocessor can detect functions from arrays (it finds all arrays in the DIMENSION statement).

8. Logical, double precision, and alphanumeric data handling is not allowed. The n-bit simulated program expects all numeric values in assignment and arithmetic statements.

9. These are certain reserved subroutine names (shown in Figure B-1) and reserved variable names which must not be used by user (except for KEY & SETNBIT). These names are used by the preprocessor.

10. No statement functions are allowed.

11. User should not use shift operations to perform multiplications or divisions (by powers of two). These operations would not be detected as arithmetic operations and would not be n-bit simulated properly.

12. Data statements should not be used for real values or values that might exceed an n-bit simulated word-length. These Data statements are not analyzed by the preprocessor.

13. Reserved labels are 79110 - 79199.

14. Only one statement per line maximum (i.e. no A=B=C=0).

RRADD	R1ADD	R2ADD	IIADD	IASGN	ITTTTAA	RTTTTAA
RRMNS	R1MNS	R2MNS	IIMNS	RASGN	ITTTTAB	RTTTTAB
RRMPY	R1MPY	R2MPY	IIMPY	KEY	:	:
RRDVD	R1DVD	R2DVD	IIDVD	SETNBIT	:	:
RREXP	R1EXP	R2EXP	IIEXP		ITTTTAZ	RTTTTAZ

Figure B-1 Reserved Variable Names For N-bit Simulated Programs

Other User Considerations:

1. Intrinsic functions (such as SIN,COS,ATAN), other FORTRAN library routines, and all other routines not read into the n-bit simulation preprocessor are not n-bit simu-

lated completely since they are coded in COMPASS and the preprocessor can only handle FORTRAN programs. However, the arguments are n-bit simulated and the resulting value from the function is n-bit simulated.

2. Array arguments are not n-bit simulated .

3. Only assignment and arithmetic statements are affected by the n-bit simulation tool. Expressions occurring elsewhere will not be affected (such as inside IF statements). Also data initialized within DATA statements will not be affected.

4. For exponentiation, n-bit simulation affects are only applied to the result of the exponentiation operation, so the results do not reflect on an n-bit wordlength machine exactly in some cases.

5. Equivalence statements are not recommended, but can be used if the programmer fully understands the impact n-bit simulation may have upon the equivalenced values.

6. Any user should be aware that any arithmetic error which would result in abnormal termination in a CDC FORTRAN program will terminate the n-bit simulated program also. Even though no abnormal termination resulted from execution in a normal CDC execution, the effects of n-bit wordlength may cause a value to be generated which could terminate it (for instance, .000001 might be truncated to 0. for a 16-bit word and would result in underflow termination when used as a divisor).

The pointers shown on the left of Figure B-2 indicate those statements which would not have appeared in the program if the program was not to be n-bit simulated.

Control Cards

The following card sequence could be used to n-bit simulate a FORTRAN programmed algorithm:

```
FTN,L=DUMMY.
LGO,,,NBIT.
REWIND,NBIT.
FTN,L=DUMMY2,B=SUBS.
REWIND,SUBS.
EDITLIB.
LIBRARY(NSUBS).
FTN,I=NBIT,B=NBITGO,L=DUMMY3.
REWIND,NBITGO.
NBITGO.
7/8/9 Card
  *** Preprocessor source program ***
7/8/9 Card
  *** USER'S SOURCE PROGRAM ***
7/8/9 Card
  *** N-bit Simulation Subroutines (Source) ***
7/8/9 Card
LIBRARY(NSUBS,NEW)
ADD(*,SUBS)
FINISH.
ENDRUN.
7/8/9 Card
6/7/8/9 End-of-Job Card
```

To n-bit simulate the USER'S SOURCE PROGRAM, the programming restrictions previously described must be strictly followed. Then, the USER'S SOURCE PROGRAM is read by the n-bit simulation preprocessor program, which transforms arithmetic statements of the original program into subroutine calls, which are designed to accomplish the n-bit simulation word-length effects. The preprocessor outputs the n-bit simulated version of the USER'S SOURCE PROGRAM. Afterwhich the n-bit

```
      PROGRAM PREPROCESSORS(INPUT,OUTPUT,TAPE1=OUTPUT)                  000100
      IMPLICIT INTEGER (A-Z)                                            000110
C     IMPLICIT INTEGERS IS NOT ALLOWED IN THE N-BIT SIMULATOR           000120
C       NOR ARE INTEGER OR REAL VARIABLE DECLARATIONS, VARIABLE         000130
C       TYPE SHOULD USE DEFAULT VALUES I-N FOR INTEGER,AND              000140
C         A-H,O-Z FOR REALS                                             000150
      DIMENSION B(2),J(16,2),K(1),KEY(7)                                000160
      COMMON A,B,DD(1)                                                  000170
      INTEGER A2,A1                                                     000180
      EQUIVALENCE (C,D),(I1,DE)                                         000190
      DATA  C/2HPR1,2Z,BLNK/1H /                                        000200
C     THIS IS  ONLY A TEST PROGRAM TO EXAMINE THE PREPROCESSORS         000210
C       HANDLING OF THE VARIOUS STATEMENT TYPES                         000220
C       A " CALL SETNBIT" MUST BE THE FIRST EXECUTABLE STATEMENT WITHIN  000230
C       EACH ROUTINE AND KEY(7) MUST BE DIMENSIONED FOR EACH ROUTINE    000240
C       IN THE N-BIT SIMULATED PROGRAM                                  000250
      CALL SETNBIT(24,1,1,1,17,6,C,KEY)                                 000260
      READ *,C6,I1                                                      000270
C       ALL EXTERNAL VALUES COMING INTO A N-BIT SIMULATED ROUTINE       000280
C       MUST BE SET EQUAL TO THEMSELVES-IN ORDER TO GIVE EACH ITS N-BIT 000290
C         SIGNIFICANCE                                                  000300
      C7=C6                                                             000310
      I1=I1                                                             000320
C     EVERY EXTERNAL VARIABLE MUST BE ASSIGNED TO ITSELF TO             000330
C       N-BIT SIMULATE ITS ACCURACY                                     000340
      READ (1,7)  (K(L),L=1,15)                                         000350
    7 FORMAT(15F3.7)                                                    000360
C       THE ARRAY K (EXTERNAL VALUES) MUST BE SET EQUAL TO ITSELF TO    000370
C         GIVE IT ITS N-BIT WORDLENGTH SIGNIFICANCE                     000380
      DO 4 L=1,15                                                       000390
      K(L)=K(L)                                                         000400
    4 CONTINUE                                                          000410
      A1=(23+3*(PRV-11.0))                                              000420
      DO 5 M=1,15                                                       000430
      C(M)=K(M)*2                                                       000440
      CONTINUE                                                          000450
      PRINT *," INPUT "                                                 000460
      PRINT *,K                                                         000470
      A1=K(1)+7.5E24/K(2)                                               000480
      CALL SUB1(25.,C,A1)                                               000490
      A=A**2/3*I+C(12)                                                  000500
      CALL MANIPUL(A,B,X,Y),RETURNS(13)                                 000510
      REWIND 1                                                          000520
      ASSIGN 13 TO ILABEL                                               000530
      DO 7 L=1,15                                                       000540
      C(L)=B(L,1)*2.5.37                                                000550
    7 CONTINUE                                                          000560
      GO TO (13,15),ILABEL                                              000570
   30 IF(I1.EQ.0)      GO TO 9                                          000580
      WRITE (1,10)  A1,(C(NN),NN=1,20,2)                                000590
   10  FORMAT(2X,=A10)                                                  000600
      GO TO 13                                                          000610
    9 PRINT *," STARTED LAST PART"                                      000620
   13 A1=C(2)*C(13)*K(.)**(-1.7)                                        000630
   15 A2=C(1)*(A1+K(.))*                                                000640
           (A1-5)+1.000                                                 000650
      STOP "  PREPROCESS TEST COMPLETE"                                 000660
      END                                                               000670
      SUBROUTINE SUB1(I1,C,A1)                                          000680
      COMMON A,B,DD(1)                                                  000690
      DIMENSION C(1),KEY(7)                                             000700
      CALL SETNBIT(24,1,1,1,17,6,C,KEY)                                 000710
```

Figure B-2 Sample Program

simulation subroutines which perform the n-bit wordlength effects are compiled and libraried. Then, the n-bit simulated user program is compiled and executed. If the program uses additional input/output files, these files would be put in their proper position along with the NBITGO card.

If the object forms of the preprocessor and n-bit simulation subroutines are used instead of the source versions, the control card sequence would be:

```
COPYBR,,SUBS,1.
INPUT,,,NBIT.
REWIND,SUBS.
EDITLIB.
LIBRARY(NSUBS).
FTN,I=NBIT,B=NBITGO,L=DUMMY3.
REWIND,NBITGO.
NBITGO.
7/8/9 Card
   ***  N-bit Simulation Subroutines (Object)  ***
7/8/9 Card
   ***  Preprocessor object deck ***
7/8/9 Card
   ***  USER'S SOURCE PROGRAM ***
7/8/9 Card
LIBRARY(NSUBS,NEW)
ADD(*,SUBS)
FINISH.
ENDRUN.
7/8/9 Card
6/7/8/9 End -of-Job Card
```

If the user wishes to see the n-bit simulated version of the USER'S SOURCE PROGRAM, the L=DUMMY3 parameter on the last FTN card should be removed. Normally this print would probably be of little interest to the user.

The user should be aware that the n-bit simulation of a program will probably increase the core requirements by four to five times and the execution time may increase up to 40 times the program's normal execution time (without n-bit simulation effects).

With the present n-bit simulation tool, values which overflow are replaced by the maximum significant value that the n-bit wordlength can represent. For example, if the resulting value of a computation exceeded the maximum negative number representable by the n-bit word, the value would be replaced by that maximum negative number. In addition, an overflow message would be printed if the user option was on. Similarly, an overflow of the positive maximum would result in the replacement of the value that overflowed with the positive maximum number. Underflow results in the replacement of zero for the value that underflowed. Same results of the n-bit simulation tool are shown in Chapter 5 of the main text.

Appendix C

Maintenance Manual For The N-bit Simulation Tool

The n-bit simulation tool consists of two parts: the preprocessor program and the n-bit simulation subroutines. This manual includes a description of the overall n-bit simulation tool system, general descriptions of the preprocessor and n-bit simulation subroutines, structure charts, interface charts, and module definitions for the preprocessor. In addition, program listings of the preprocessor and the n-bit simulation subroutines are included.

Overall N-bit Simulation Tool

The preprocessor reads a FORTRAN program (to be n-bit simulated) and converts each arithmetic statement into one or more subroutine calls. A subroutine call is generated for each arithmetic operation of an arithmetic statement. The subroutine calls,generated by the preprocessor,reference the n-bit simulation subroutines. When the n-bit simulated version of the original algorithm is executed, the n-bit simulation subroutines perform the n-bit wordlength effects upon each arithmetic statement operation.

The n-bit wordlength effects are specified by the user within each routine of the original FORTRAN program. The results from the execution of the n-bit simulated version are nearly the exact result obtainable from a computer of n-bit wordlength.

Preprocessor

The preprocessor basically reads and syntactically analyzes each statement of the user's program. Each arithmetic statement is transformed into a Polish Notation string, then decomposed one operation at a time. For each operation, a subroutine call is generated. The subroutine referenced will perform the arithmetic computation indicated by the subroutine name. Each combination of variable data types is also handled by a separate subroutine. The operands which the operation is being performed upon may be either integer or real. A different subroutine reference is generated for each possible combination of operand data types and arithmetic operation.

The module structure of the preprocessor is shown in Figure C-1. The module definitions follow, along with a chart describing the preprocessor's module interfaces. The program listing of the preprocessor follow Figure C-1.

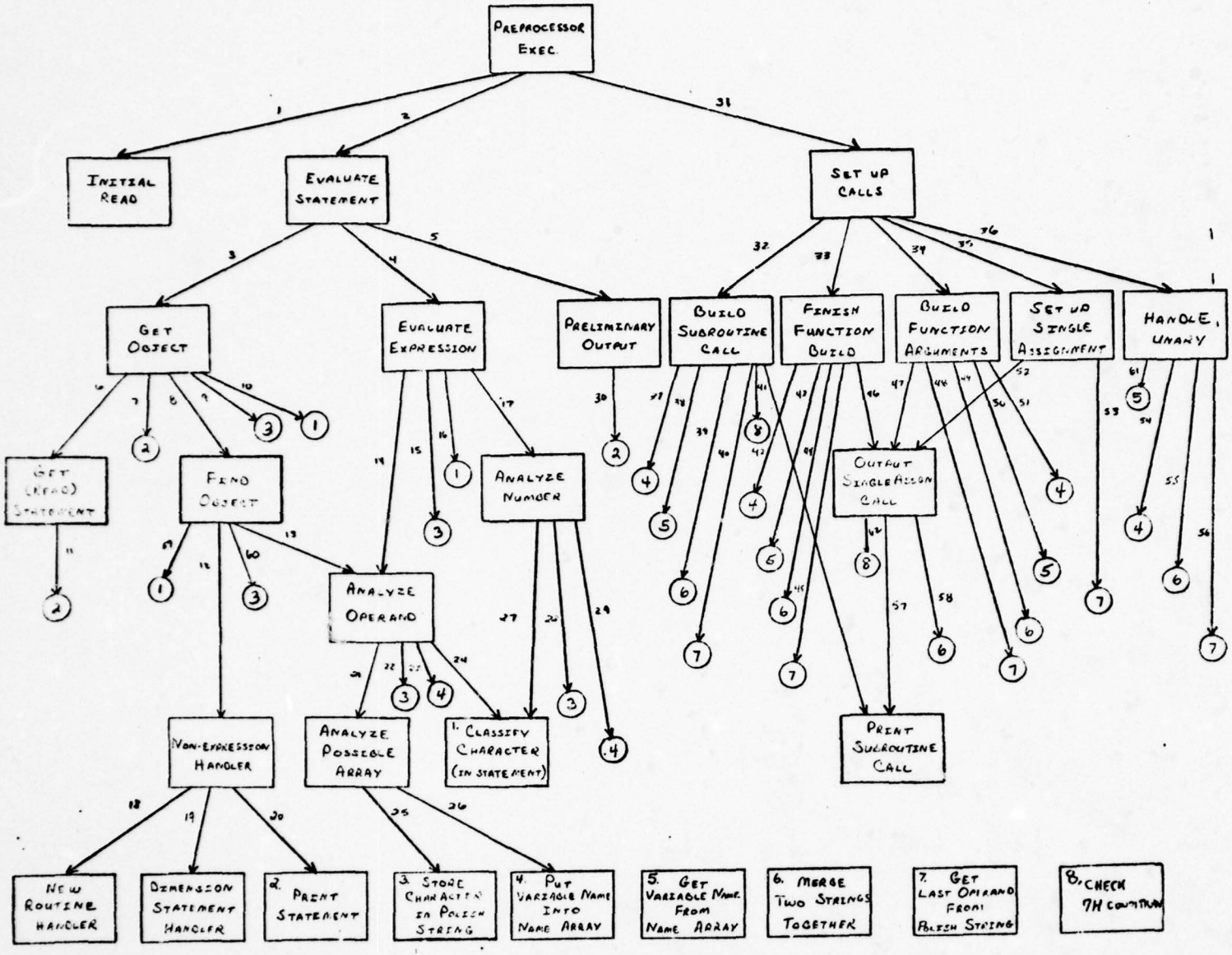


Figure C-1 Preprocessor Program Strucure

Module Definitions And Module Interfaces For The Preprocessor

PREPROCESSOR-EXEC is the main routine of the preprocessor program. It calls the afferent and efferent branches (EVALUATE-STATEMENT and SET-UP-CALLS) which carry out the translation of a FORTRAN algorithm to the n-bit simulation equivalent algorithm in addition to the INITIAL-READ module.

INITIAL-PEAD does the initial read of the FORTRAN algorithm. It reads the contents of the first program card into a temporary array which is used by the GET-STATEMENT module, which functions on a 'look-ahead' basis.

EVALUATE-STATEMENT supervises the evaluation of (legal) arithmetic statements into their Polish Notation equivalent representation.

GET-OBJECT supervises the check of a statement for a legal object, where an arithmetic statement structure consists of object = arithmetic expression .

GET-STATEMENT reads input program's cards, passing a 'statement' a time to its superordinates, GET-OBJECT. A statement may stretch over several cards through the use of continuation cards, so a look-ahead read is performed.

FIND-OBJECT determines if a statement has a legal object.

NON-EXPRESSION-HANDLER calls for the print of a non-expression statement and determines if the non-expression is a key statement type, for dimensioning arrays or the beginning of a new routine.

NEW-ROUTINE-HANDLER extracts the program unit's name and initializes the line counter for that new unit to one. (This information is used for each arithmetic function subroutine call built by the efferent branch). Also it clears the list of variable names declared as arrays for the previous program unit.

DIMENSION-STATEMENT-HANDLER stores all variable names which have been declared arrays.

PRINT-STATEMENT prints the statement which may span several cards to the n-bit simulated program output file.

EVALUATE-EXPRESSION is a recursive module which determines (through a syntax analysis) if the expression is a legal arithmetic expression, and converts the statement into its Polish Notation equivalent representation. It is recursive because parentheses may be used to enclose another complete expression and parentheses may occur throughout an expression.

ANALYZE-OPERAND determines if the syntax of a string of characteristics constitiutes a legal variable name (a variable may also be an array or function reference).

ANALYZE-NUMBER determines if the syntax of a string of characters constitutes a legal number.

ANALYZE-POSSIBLE-ARRAY determines if a variable name has been declared an array.

CLASSIFY-CHARACTER types input character as a letter, digit, operator, parenthesis, decimal point,or comma.

STORE-CHARACTER-IN-POLISH-STRING stores a variable name or operator in Polish string according to Reverse Polish Notation and the operator precedence.

PUT-VARIABLE-NAME stores a string of characters in the NAME array and passes back to superordinate a pointer to the position of the string in the NAME array.

PRELIMINARY-OUTPUT outputs a COMMENT statement containing the input statement and outputs preliminary output for labels and IF statements.

SET-UP-CALLS scans Polish string left to right and calls a handling routine for each operator found.

BUILD-SUB-CALLS assembles and merges parts of a subroutine call.

PRINT-SUB-CALL prints 80 character card images to the n-bit simulated program output file.

BUILD-FUNC-ARGUMENTS merges function arguments together, generating calls for single arguments if necessary.

FINISH-FUNCTION merges the function name with its argument list (built by BUILD-FUNC-ARGUMENTS) and generates a single assignment call for the function.

SET-UP-SINGLE-ASSIGNMENT sets up parts for a single assignment.

HANDLE-UNARY places a unary sign in front of the variable name through a call to the PUT-VARIABLE-NAME module.

OUTPUT-SINGLE-ASSIGNMENT merges parts required for a single assignment function call.

GET-VARIABLE-NAME gets a string of characters from the name array according to the pointer given and passes to the caller the string, its data type, and its length.

MERGE merges strings of characters together.

Module Interfaces

Interface # (Reference Fig. 29)	In	Out
1,2,31	-	-
3	STMT,POSITION	-
4,5	-	STMT,POSITION
6	-	STMT
7,11,20,30	-	STMT
8	-	STMT,POSITION
9,15,22,25,28, 60	-	CHARACTER
10,16,24,27,59	POSITION	STMT
12	-	STMT
13,14	POSITION	STMT,POSITION,TYPE
17	POSITION	STMT,POSITION,TYPE
18	-	STMT,POSITION
19	-	STMT,POSITION
21	POSITION	STMT,POSITION,NAME, LENGTH,NMPOINTER, MODE
23,26,29,37,42, 51,54	NAMEPOSITION	NAME,LENGTH,MODE
32	TEMPNAMES	POLISH,POLPOSITION, TEMPNAMES,FINALFLG, TRLR,TRLRLEN
33	TEMPNAMES	POLISH,POLPOSITION, TEMPNAMES,FINALFLG, TRLR,TRLRLEN
34	-	POLISH,POLPOSITION, TEMPNAMES,TRLR, TRLRLEN
35	-	POLISH,POLPOSITION, TRLR,TRLRLEN
36	-	POLISH,POLPOSITION
38,44,50,61	NAME,MODE,LENGTH	NAMEPOSITION
39,44,49,55,58	ARRAY1,LEN1	ARRAY1,LEN1,NAME2, LEN2
40,45,48,53,56	OPRNDPOSITION	POLISH,POLPOSITION
41,57	-	OUT,LENGTH
46,47,52	-	NAME,NAMLEN,MODE,OBJCT, OBJLEN,TRLR,TRLRLEN

N-bit Simulation Preprocessor

Program Listing

```
      PROGRAM PREPOS(INPUT,OUTPUT,TAPE1=4002/80,TAPE2=INPUT)                 000100
C                                                                            000102
C       N-BIT SIMULATION PREPROCESSOR PROGRAM WILL CONVERT AN                000103
C        INPUT FORTRAN PROGRAM INTO ITS EQUIVALENT N-BIT SIMULATED           000104
C        VERSION WHICH USES SUBROUTINE CALLS TO PERFORM THE N-BIT            000105
C        WORDLENGTH EFFECTS UPON THE INPUT PROGRAM'S ARITHMETIC STMTS         000106
C                                                                            000109
      IMPLICIT INTEGER (A-Z)                                                 000110
C      TO THE INITIAL READ OF THE INPUT PROGRAM                              000118
      CALL INITPO,RETURNS(100)                                               000120
C         BUILD POLISH STRING FOR EACH ARITHMETIC STATEMENT                  000123
C           ALT RETURN IS USED WHEN AN EOF HAS BEEN ENCOUNTERED              000124
    3 CALL EVLSTMT,RETURNS(100)                                              000130
C      OUTPUT N-BIT SIMULATED CODE FOR EACH ARITHMETIC STATEMENT             000134
C       WHICH IS NOW EXPRESSED IN POLISH NOTATION                            000135
      CALL SETUP                                                             000140
      GO TO 3                                                                000150
C      EOF ENCOUNTERED                                                       000157
  100 STOP " EVLSTMT COMPLETED"                                              000160
      END                                                                    000170
      SUBROUTINE EVLSTMT,RETURNS(N1)                                         000180
      IMPLICIT INTEGER(A-Z)                                                  000190
C     EVLSTMT SUPERVISES THE EVALUATION OF LEGAL ARITHMETIC                  000192
C         STATEMENTS INTO THEIR POLISH NOTATION EQUIVALENT                   000193
C               REPRESENTATION                                               000194
      DIMENSION STMT(403)                                                    000200
    5 CALL GETOBJ(STMT,0),RETURNS(100)                                       000210
      CALL EVLEXP(STMT,0),RETURNS(5)                                         000220
C       PRELIMINARY OUTPUT FOR THE ARITHMETIC STATEMENT FOR SUCH             000224
C        THINGS AS LABELS AND IF STMTS IN ADDITION TO PRINTING A             000225
C        A COMMENT STATEMENT CONTAINING THE ORIGINAL STATEMENT               000226
      CALL PRELOUT(STMT)                                                     000230
      RETURN                                                                 000240
  100 RETURN N1                                                              000250
      END                                                                    000270
      SUBROUTINE EVLEXP(STMT,P),RETURNS(N2)                                  000280
      IMPLICIT INTEGER(A-Z)                                                  000284
C                                                                            000285
C          N2 ALTERNATE RETURN USED AFTER THE DETECTION AND                  000286
C           AND REPORTING OF AN ILLEGAL ARITH STATEMENT                      000287
C                                                                            000290
C     EVALUATE-EXPRESSION IS A RECURSIVE MODULE WHICH                        000291
C       DETERMINES(THROUGH SYNTAX ANALYSIS) IF THE EXPRESSION                000292
C       IS A LEGAL ARITHMETIC EXPRESSION,AND CONVERTS THE STATE-             000293
C       MENT INTO ITS POLISH NOTATION EQUIVALENT REPRESENTATION              000294
C       IT IS RECURSIVE BECAUSE PARENTHESES MAY BE USED TO                   000295
C       ENCLOSE ANOTHER COMPLETE EXPRESSION,WHICH MUST UNDERGO               000296
C       EXACTLY THE SAME SYNTAX ANALYSIS. PARENTHESES MAY OCCUR              000297
C       THROUGHOUT AN EXPRESSION.                                            000298
C                                                                            000299
      DEFINITION STMT(1)                                                     000300
      DATA UNARY/101/,TIMES/102/,EXP/103/,PLUS/104/,MINUS/105/,FUNC/122/     000320
C     FUNCFLG IS A FLAG INDICATING WHETHER A FUNCTION IS BEING EVALUATED     000330
      FUNCFLG=0                                                              000340
      NEG=0                                                                  000350
C     NEG, INDICATES IF IS VALID TO HAVE AN EXPRESSION                       000360
    1 NEG=1                                                                  000370
      CALL CLASIFY(STMT,P,TYPE),RETURNS(5,200)                               000380
C     --- CHECK FOR UNARY + OR - (STMT(P)=105)                               000390
      IF(TYPE.NE.5)  GO TO 2                                                 000410
C     UNARY PLUS OR MINUS IS IGNORED, SINCE THEY HAVE NO REAL MEANING        000417
C     ONLY UNARY MINUS IS USED IN CONVERSION TO THE POLISH STRING            000418
```

```
      IF(STMT(P).EQ.PLUS)     GO TO 40                                  000420
      IF(STMT(P).NE.MINUS)    GO TO 600                                 000430
      CALL STPOL(UNARY)                                                 000440
C   LOOKING FOR AN OPERAND                                              000450
 40   CALL CLASIFY(STMT,P,TYPE),RETURNS(500,600)                        000460
C     -- LOOKING FOR THE FIRST VALID CHARACTER OF A STRING              000470
C       LETTER # OPERATOR ( ) , . = OTHER                               000480
 45   GO TO (50,70,,30,110,600,600,70,600,600),TYPE                     000490
C POSSIBLE OPERAND                                                      000500
 50   OK=0                                                              000510
      CALL OPRND(STMT,P,TYPE),RETURNS(530,600,55)                       000520
C   ABOVE ALT RETURNS EOS,ERROR,FUNCTION POSSIBILITY                    000530
C       THE FOLLOWING CHECK FOR LEGAL FOLLOWERS OF OPERANDS             000534
C      CHECK FOR OPERATOR                                               000535
      IF(TYPE.EQ.7)  GO TO 90                                           000540
C   --- CHECK FOR RIGHT PAREN ---                                       000550
      IF(TYPE.EQ.5)   GO TO 130                                         000560
C      --CHECK FOR COMMA--                                              000570
      IF(TYPE.EQ.6)  GO TO 150                                          000580
C         OTHERWISE ILLEGAL FOLLOWER OF AN OPERAND                      000584
      GO TO 600                                                         000590
C    --- GOT A POSSIBLE FUNCTION --- SET FUNCTION FLAG                  000600
 55   CALL STPOL(FUNID)                                                 000610
      NPAREN=NPAREN+1                                                   000620
      FNCFLG=1                                                          000630
      OK=1                                                              000640
C      RECURSE TO THE BEGINNING OF ROUTINE TO EVALUATE EXPRESSIONS      000645
C        USED AS FUNCTION ARGUMENTS                                     000646
      GO TO 1                                                           000650
C     GOT A  POSSIBLE #                                                 000660
 70   OK=0                                                              000670
      CALL NUM(STMT,P,TYPE),RETURNS(500,600)                            000680
      IF(TYPE.EQ.7)   GO TO 90                                          000690
      IF(TYPE.EQ.5)   GO TO 170                                         000700
      IF(TYPE.EQ.6)   GO TO 150                                         000710
C   ILLEGAL FOLLOWER OF #                                               000720
      GO TO 600                                                         000730
C     -- GOT AN OPERATOR                                               000740
 90   OK=1                                                              000750
C      CHECK FOR POSSIBLE EXPONENTIATION                                000755
      IF(STMT(P).EQ.TIMES)    GO TO 93                                  000760
C      STORE OPERATOR IN POLISH STRING                                  000764
 91   CALL STPOL(STMT(P))                                               000770
C      --- GO LOOK FOR AN OPERAND                                       000780
      GO TO 40                                                          000800
C     ---- CHECK FOR EXPONENTIATION  -----                              000810
 93   LP=P                                                              000820
      CALL CLASIFY(STMT,LP,TYPE),RETURNS(600,600)                       000830
      IF(STMT(LP).NE.TIMES)    GO TO 91                                 000840
C       -- SET LP UP --                                                 000850
      P=LP                                                              000860
C  --       STORE EXPONENTIATION IN POLISH STRING      --               000870
      CALL STPOL(EXP)                                                   000880
      GO TO 40                                                          000890
C     --- GOT LEFT PAREN  - START OF A NEW EXPRESSION --               000900
 110  NPAREN=NPAREN+1                                                   000910
      CALL STPOL(STMT(P))                                               000920
C     ---- NEED TO RECURSE TO SET OK TO 1                               000930
      OK=1                                                              000940
      GO TO 1                                                           000950
C     -- GOT A RIGHT PAREN  --                                          000960
 130  NPAREN=NPAREN-1                                                   000970
      CALL STPOL(STMT(P))                                               000980
C     ---- NEED TO CHECK TO SEE IF IT WAS A LEGAL PARENTHESIZED EXPRESSION 000990
      IF(OK.NE.0)     GO TO 91                                          001000
C     ---- NEXT CHARACTER MUST BE SET TO SEE IF IT A LEGAL FOLLOWER OF   001010
```

```
C         RIGHT PARENTHESIS                                              001024
      CALL CLASIFY(STMT,P,TYPE),RETURNS(500,600)                         001030
      IF(TYPE.EQ.7)      GO TO 90                                        001040
      IF(TYPE.EQ.5)     GO TO 130                                        001050
      IF(TYPE.EQ.6)      GO TO 150                                       001060
      GO TO 600                                                         001070
C     GOT A COMMA                                                       001080
  150 OK=1                                                              001090
C     SHOULD NOT HAVE A COMMA UNLESS IT IS INSIDE A FUNCTION            001100
      IF(FNCFLG.EQ.0)     GO TO 600                                     001110
      CALL STPOL(STMT(P))                                               001120
C      NEED AN OPERAND                                                  001130
      GO TO 1                                                           001140
C CHECK FOR END                                                         001150
C     CHECK FOR ENDING PROPERLY                                         001160
  500   IF(OK.NE.0.OR.NPAREN.NE.0)     GO TO 600                      001170
C    NOW CLEAR STACK IN POLISH STORE ROUTINE                            001180
      CALL STPOL(STMT(P))                                               001190
      RETURN                                                            001200
C      ILLEGAL ARITHMETIC EXPRESSION, SO PRINT IT                       001210
  600 CALL PRNT(STMT)                                                   001216
      RETURN N2                                                         001220
      END                                                               001230
      SUBROUTINE OPRND(STMT,P,TYPE),RETURNS(N1,N2,N3)                   001240
      IMPLICIT INTEGER (A-Z)                                            001250
C                                                                       001256
C         OPRND ANALYZES A POSSIBLE OPERAND TO DETERMINE IF THE         001260
C          SYNTAX OF A STRING OF CHARACTERS CONSTITUTES A LEGAL         001261
C          VARIABLE NAME (A VARIABLE MAY ALSO BE AN ARRAY OR            001262
C          FUNCTION REFERENCE).                                         001263
C        OPRND CHECKS TO SEE IF CHAR SEQUENCE IS A VALID OPERAND        001264
C      IT LIMITS THE SEQUENCE TO 7 CHAR MAX AND WILL DETERMINE THE MODE 001270
C                                                                       001280
      DIMENSION STMT(1)                                                 001289
      DATA I/1HI/,N/1HN/,ZERO/1H0/,BLANKS/1H           /,FUNC/1HF/      001290
      DATA MAXLEN/7/                                                    001300
C     TESTING TO SEE IF FIRST CHARACTER OF OPERAND IS A LETTER          001310
      IF(TYPE.NE.1)   GO TO 400                                         001320
C    MODE=1(REALS)    =2 (INTEGERS)                                     001330
      MODE=1                                                            001340
C       CHECK FOR INTEGER VARIABLE(BASED ON LEADING CHARACTER)          001350
      IF(STMT(P).GE.I.AND.STMT(P).LE.N)  MODE=2                         001360
      CNT=0                                                             001370
      NAME=BLANKS                                                       001380
C       CNT COUNTS # CHARACTERS IN VARIABLE NAME                        001381
C        BELOW COMPUTED # BITS CHARACTER BEING STORED MUST BE           001382
C       SHIFTED TO PLACE IN PROPER POSITION WITH RESPECT TO             001383
C        THE OTHER CHARACTERS ALREADY STORED FROM THE VARIABLE NAME     001384
C     LEFT=(3-CNT)*6                                                    001385
C     CHANGE TO SAVE BLANKS SOMETIME**********************************  001390
      NAME=SHIFT(STMT(P),LEFT).OR.(MASK(CNT*6).AND.NAME)                001400
      CNT=CNT+1                                                         001410
      CALL CLASIFY(STMT,P,TYPE),RETURNS(500,600)                        001420
C       TYPE > 2 INDICATES NEXT CHARACTER IS NOT LETTER OR DIGIT        001430
      IF(TYPE.GT.2)  GO TO 35                                           001440
C      MAX LENGTH OF A VARIABLE NAME IS SEVEN                           001450
      IF(CNT.EQ.7) GO TO 500                                            001460
      GO TO 6                                                           001470
C      GOT POSSIBLE ARRAY/FUNCTION(TYPE = INDICATES LEFT PAREN)         001480
   35 IF(TYPE.EQ.4)  GO TO 45                                           001490
C    END OF VARIABLE- GO STORE IT AWAY                                  001500
      CALL PUTNAME(NAME,CNT,MODE,PTR)                                   001510
      CALL STPOL(PTR)                                                   001520
      GO TO 23                                                          001530
C       RETURN (N1) INDICATES NAME WAS NOT DECLARED AN ARRAY-           001533
C        SO IT IS A VARIABLE FUNCTION                                   001534
```

```
C          THEN CLASSIFY NEXT CHARACTER FOLLOWING ARRAY                    001525
C            ALT RETURN (48) INDICATES EOS,WHICH IS LEGAL END              001526
C            TO AN EXPRESSION                                              001527
 45    CALL PROARRY(STMT,P,NAME,CNT-1,NPTR,MODE),RETURNS(48)                001530
       CALL CLASIFY(STMT,P,TYPE),RETURNS(500,400)                           001540
       GO TO 53                                                             001550
C   GOT A FUNCTION                                                         001560
 48    CALL PUTNAME(NAME,CNT,MODE,PTR)                                      001570
       CALL STPOL(PTR)                                                      001580
C          RETURN N3 INDICATES OPERAND WAS A FUNCTION                      001590
       RETURN N3                                                            001600
 53    RETURN                                                               001610
C          RETURN N2 -ILLEGAL CHARACTER WAS ENCOUNTERED(THUS AN            001612
C           AN ILLEGAL OPERAND)                                            001614
 400   RETURN N2                                                            001620
 500   CALL PUTNAME(NAME,CNT,MODE,PTR)                                      001630
C          EOS WAS ENCOUNTERED SO STORE VARIABLE NAME                      001632
       CALL STPOL(PTR)                                                      001640
 530   RETURN N1                                                            001650
       END                                                                  001660
       SUBROUTINE NUM(STMT,P,TYPE),RETURNS(N1,N2)                           001670
       IMPLICIT INTEGER (A-Z)                                               001680
C          ALT RETURN N1=EOS,N2=ILLEGAL #                                  001682
C          NUM DETERMINES IF THE SYNTAX OF A STRING OF CHARACTERS          001683
C           CONSTITUTES A LEGAL NUMBER                                     001684
C     CHECK FOR LEGAL INTEGER/REAL #S                                      001690
       DIMENSION STMT(1),NUM2(3)                                            001700
       DATA DECPT/1H./,BLANKS/10H          /                             001710
C  DEFAULT IS INTEGER (MODE=2)                                             001720
       MODE=2                                                               001730
C        IF 1ST CHAR IS A DECIMAL PT, SET MODE TO 1(FOR REALS)             001732
       IF(STMT(P).EQ.DECPT)  MODE=1                                         001740
       IF(TYPE.NE.2.AND.TYPE.NE.7)    GO TO 600                             001750
       N=1                                                                  001760
C  *******MAY WANT TO CHECK FIRST CHAR IN FOR A NUMBER*****************     001770
       CNT=0                                                                001780
C        SHIFT CHARACTER INTO STORAGE POSITION                             001785
 10    LEFT=(6-CNT)*6                                                       001790
       NUM2(N)=SHIFT(STMT(P),LEFT).OR.(MASK(CNT*6).AND.NUM2(N))             001800
       CNT=CNT+1                                                            001810
       CALL CLASIFY(STMT,P,TYPE),RETURNS(500,600)                           001820
       IF(TYPE.EQ.2)    GO TO 20                                            001830
C        ---CHECK FOR DECIMAL POINT------                                  001840
       IF(TYPE.EQ.7)  GO TO 15                                              001850
C        --- CHECK FOR OPERATOR ---                                        001860
       IF(TYPE.EQ.3)    GO TO 53                                            001870
C   ---   ALSO A ) OR , IS A LEGAL END TO A NUMBER  ----------------       001880
       IF(TYPE.EQ.4.OR.TYPE.EQ.5)   GO TO 53                                001890
       GO TO 600                                                            001900
 15    IF(MODE.EQ.1)  GO TO 600                                             001910
       MODE=1                                                               001920
 20    IF(CNT.LT.10)    GO TO 10                                            001930
C          INCREMENT TO NEXT WORD,IF 1ST WORD IS FULL                      001935
       N=N+1                                                                001940
       CNT=0                                                                001950
       GO TO 10                                                             001960
 50    CNT=(N-1)*10+CNT                                                     001970
       CALL PUTNAME(NUM2,CNT,MODE,PTR)                                      001980
       CALL STPOL(PTR)                                                      002000
       RETURN                                                               002010
C          COMPUTE LENGTH OF THE # THEN RETURN FOR EOS CONDITION           002015
 55    CNT=(N-1)*10+CNT                                                     002020
       CALL PUTNAME(NUM2,CNT,MODE,PTR)                                      002030
       CALL STPOL(PTR)                                                      002040
       RETURN N1                                                            002050
C     ---- ILLEGAL NUMBER RETURN ------                                    002060
```

```
 600  RETURN N2                                                              002070
      END                                                                    002080
      SUBROUTINE PROARRY(STMT,P,NAME,NLEN,NPTR,TYPE),RETURNS(N1)             002090
C                                                                            002098
C     ALT RETURN N1 INDICATES A THE VARIABLE WAS NOT DIMENSIONED-            002100
C          --- PROBABLE FUNCTION                                            002101
C        PROARRY DETERMINES IF A VARIABLE NAME HAS BEEN                      002105
C         DECLARED IN ARRAY,IF SO IT STORES THE ARRAY NAME AND ITS           002107
C         ARGUMENT LIST AWAY IN THE NAME ARRAY(VIA PUTNAME)                  002108
C                                                                            002109
      IMPLICIT INTEGER (A-Z)                                                 002110
      COMMON/ARRAYS/ARRAY(400),ARRYMAX                                       002120
      DIMENSION STMT(1),ANAME(5)                                             002130
      DATA LPAREN/1H(/,RPAREN/1H)/,BLNK/1H /,EOS/3REOS/,MAXALEN/5/           002140
      IF(ARRYMAX.EQ.0)   GO TO 6                                             002150
      DO 5 I=1,ARRYMAX                                                       002160
      IF(NAME.EQ.ARRAY(I))  GO TO 14                                         002170
 5    CONTINUE                                                               002180
C GOT A FUNCTION PROBABLY                                                    002190
 6    RETURN N1                                                              002200
C STARTING TO BUILD ARRAY ITEM AND (                                         002210
 14   ANAME(1)=NAME                                                          002220
      ACNT=1                                                                 002230
      PARCNT=1                                                               002240
 17   NLEN=NLEN+1                                                            002250
 18   LEFT=(3-NLEN)*6                                                        002260
      ANAME(ACNT)=SHIFT(STMT(P),LEFT).OR.(MASK(NLEN*6).AND.ANAME(ACNT))      002270
C      IF MATCHED RPAREN FOUND, THE ARRAY IS COMPLETE-GO ON                  002274
      IF(PARCNT.EQ.0)     GO TO 500                                          002280
C       STORE AWAY REST OF ARRAY (UNTIL MATCHING RPAREN FOUND)               002285
 19   P=P+1                                                                  002290
      IF(STMT(P).EQ.BLNK)    GO TO 19                                        002300
      IF(STMT(P).EQ.EOS)    GO TO 500                                        002310
      IF(STMT(P).EQ.RPAREN)  PARCNT=PARCNT-1                                 002320
      IF(STMT(P).EQ.LPAREN)  PARCNT=PARCNT+1                                 002330
      IF(NLEN.LT.3)      GO TO 17                                            002340
C        INCREMENT TO NEXT ARRAY STORAGE WORD AND ZERO THE COUNTER           002345
C              FOR LENGTH                                                    002346
      IF(ACNT.EQ.MAXALEN)    GO TO 530                                       002360
      ACNT=ACNT+1                                                            002370
      NLEN=0                                                                 002380
      GO TO 18                                                               002390
C      COMPUTE ARRAY LENGTH AND STORE IT ALL                                 002395
 500  TLEN=10*(ACNT-1)+NLEN+1                                                002400
      CALL PUTNAME(ANAME,TLEN,TYPE,NPTR)                                     002410
      CALL STPOL(NPTR)                                                       002420
      RETURN                                                                 002430
      END                                                                    002440
      SUBROUTINE CLASIFY (STMT, P, TYPE), RETURNS(N1, N2)                    002450
C     N1=EOS, N2=ILLEGAL                                                     002460
C      CLASIFY TYPES INPUT NEXT CHARACTER IN STMT ARRAY                      002461
C      AS A LETTER,DIGIT,OPERATOR,PARENTHESIS,DECIMAL POINT,OR COMMA         002462
C        BLANKS WITHIN A STMT ARE IGNORED                                    002463
      IMPLICIT INTEGER (A-Z)                                                 002470
      DIMENSION STMT(1), CHAR(64)                                            002480
      DATA EOS/3REOS/,CHAR/0,28*1,1*2,4*3,-,.,/,8,10,6,7,11*9/,              002490
         LAST/12*/,BLNK/1H /                                                 002500
C                                                                            002509
C     TYPE=1 FOR LETTERS                                                     002510
C     TYPE=2 FOR NUMBERS                                                     002511
C     TYPE=3 FOR OPERATORS                                                   002512
C     TYPE=4 FOR (                                                           002513
C     TYPE=5 FOR )                                                           002514
C     TYPE=6 FOR ,                                                           002515
C     TYPE=7 FOR .                                                           002516
C     TYPE=8 FOR =                                                           002517
```

```
C     TYPE=3 FOR OTHER (ILLEGAL CHARACTERS FOR AN ARITHMETIC STMT)      002518
C                                                                       002519
5     P=P+1                                                             002520
      IF(STMT(P).EQ.EOS)  RETURN N1                                     002530
C     I(_AST) IS THE MAXIMUM ASCII VALUE ALLOWED IN PREPROCESSOR        002533
      IF(STMT(P).GT.LAST)  GO TO10                                      002540
C         BLANKS ARE IGNORED                                          002543
      IF(STMT(P).EQ.BLNK)   GO TO 5                                     002550
      TYPE=CHAR(STMT(P)+1)                                              002550
C      CHECK FOR ILLEGAL CHARACTER                                      002570
      IF(TYPE.EQ.3)  GO TO 10                                           002580
      RETURN                                                            002590
10    TYPE=3                                                            002600
      RETURN N2                                                         002610
      END                                                               002620
      SUBROUTINE STPOL(CHAR)                                            002630
      IMPLICIT INTEGER (A-Z)                                            002640
C       STPOL STORES A VARIABLE NAME PTR ( A LETTER) OR OPERATOR        002642
C       IN POLISH STRING ACCORDING TO REVERSE POLISH NOTATION AND       002643
C       USING THE OPERATOR PRECEDENCE ESTABLISHED.                      002644
C     ROUTE ERROR MESSAGES TO A SEPARATE FILE (NOT THE ONE TO BE        002650
C          COMPILED)                                                    002660
C     PUT BOS AT BEGINNING, EOS AT END                                  002670
      DIMENSION STACK(30,2),OPS(2,9)                                    002680
      COMMON/POLISHS/NPOL,NSTK                                          002690
C        COMMON/POL/POLISH(60,2),NPOL???????LATER NOW JUSST DIMENSION** 002700
      COMMON/POL/POLISH(60,2)                                           002710
      DATA STKMAX/30/,POLMAX/60/,LPAREN/1H(/,RPAREN/1H)/,EOS/3REOS/     002720
C   I= EXPONENTIATION !=UNARY  ?=FUNCTION   ,=FUNCTION ARGUMENT SEPARATOR002730
      DATA OPS/1H+,-1,1H-,-1,1H*,-2,1H/,-2,1H!,-3,                      002740
C        FIRST CHAR MUST BE A # OR .                                    002743
     *  1H,,-12,1H?,-13,1H=,-13/                                        002750
      DATA LPRNK/-1/,BOS/-50/,STACK(1,1)/-50/,FUNC/1H?/,ALPHA/3F/       002760
      DATA NSTK/1/,NPOL/1/                                              002770
      DATA IFTYPE/1HI/,POLISH(1,1)/3RBOS/                               002780
C     NSTK & NPOL HAVE TO BE INITIALIZED SOMEWHERE                      002790
C     ********POLISH(1,1)=BOS ALWAYS,REINTIALIZED TO NPOL=1 EACH TIME  002820
      IF (CHAR.EQ.EOS) GO TO 15                                         002830
C     IF CHAR > ALPHA--ITS NOT AN ALPHANUMERIC CHARACTER (OR OPERAND)   002840
      IF(CHAR.EQ.IFTYPE)    GO TO 16                                    002850
      IF(CHAR.GT.3F)    GO TO 19                                        002860
C     GOT AN OPERAND                                                    002870
C     POLMAX - MAXIMUM LENGTH OF POLISH ARRAY                           002874
16    IF(NPOL.GE.POLMAX)   GO TO 900                                    002880
      NPOL=NPOL+1                                                       002890
      POLISH(NPOL,1)=CHAR                                               002900
      POLISH(NPOL,2)=0                                                  002910
      GO TO 100                                                         002920
19    IF (CHAR.EQ.LPAREN) GO TO 40                                      002930
      IF (CHAR.EQ.RPAREN) GO TO 50                                      002940
C       TREAT FUNCTION PAREN LIKE A REGULAR PARENTHESIS                 002950
      IF(CHAR.EQ.FUNC)   GO TO 40                                       002960
C     GOT AN OPERATOR                                                   002970
      DO 30 I=1,9                                                       002980
      IF (CHAR.EQ.OPS (1,I)) GO TO 23                                   002990
30    CONTINUE                                                          003000
      PRINT *,"ERROR-ILLEGAL OPERATION"                                 003020
      STOP "BAD OP"                                                     003030
C     MATCHED AN OPERATOR                                               003040
23    IF(OPS(2,I).GT.STACK(NSTK,1)) GO TO 35                            003050
      IF (NPOL.GT.POLMAX) GO TO 900                                     003060
      NPOL=NPOL+1                                                       003070
C     NEGATIVE VALUES FOR OPERATORS ARE STORED SO THAT THEY CAN         003075
C          EASILY BE DISTINGUISHED FROM OPERAND NAMES LATER WHEN        003076
C          IS DECOMPILED                                                003077
      POLISH(NPOL,1)=-STACK(NSTK,2)                                     003080
```

```
      POLISH(NPOL,2)=0                                                  003090
      NSTK=NSTK-1                                                       003100
      GO TO 23                                                          003110
C     STORE AWAY THIS OPERATOR IN THE STACK                             003120
35    IF (NSTK.GE.STKMAX) GO TO 500                                     003130
      NSTK=NSTK+1                                                       003140
      STACK (NSTK,1)=OPS(2,I)                                           003150
      STACK (NSTK,2)=CHAR                                               003160
      GO TO 100                                                         003170
C     GOT LEFT PAREN                                                    003180
40    IF (NSTK.GE.STKMAX) GO TO 500                                     003190
      NSTK=NSTK+1                                                       003200
      STACK (NSTK,1)=LPRNK                                              003210
C     LPRNK SETS THE LEFT PREN RANK PRECEDENCE INTO STACK               003213
      STACK(NSTK,2)=CHAR                                                003220
      GO TO 100                                                         003230
C     GOT RIGHT PAREN                                                   003240
50    IF (NSTK.EQ.1) GO TO 500                                          003250
      IF(STACK(NSTK,2).EQ.LPAREN) GO TO 54                              003260
C     REACT END OF A FUNCTION TOO                                       003270
      IF (NPOL.GT.POLMAX) GO TO 500                                     003280
      NPOL=NPOL+1                                                       003290
      POLISH(NPOL,1)=-STACK(NSTK,2)                                     003300
      POLISH(NPOL,2)=0                                                  003310
C  ----IF IT WAS A FUNCTION INDICATOR RETURN,INDICATOR ADDED TO POLISH  003320
      IF(STACK(NSTK,2).EQ.FUNC)   GO TO 54                              003330
      NSTK=NSTK-1                                                       003340
C  ----IF IT WAS A FUNCTION INDICATOR RETURN,INDICATOR ADDED TO POLISH  003350
      GO TO 50                                                          003360
C    NOW REMOVE LEFT PAREN OR FUNCTION FROM STACK                       003370
 54   NSTK=NSTK-1                                                       003380
      GO TO 100                                                         003390
C      EOS PROCESSING-- CLEAR OPERATOR STACK                            003400
 75   IF(STACK(NSTK,1).EQ.EOS)   GO TO 90                               003410
      IF (STACK (NSTK,2).EQ.LPAREN) GO TO 500                           003420
      IF (NPOL.GT.POLMAX) GO TO 500                                     003430
      NPOL=NPOL+1                                                       003440
      POLISH(NPOL,1)=-STACK(NSTK,2)                                     003450
      POLISH(NPOL,2)=0                                                  003460
      NSTK=NSTK-1                                                       003470
      GO TO 75                                                          003480
 90   NPOL=NPOL+1                                                       003490
      IF(NPOL.GT.POLMAX)    GO TO 500                                   003500
      POLISH(NPOL,1)=EOS                                                003510
      POLISH(NPOL,2)=0                                                  003520
 100  CONTINUE                                                          003530
      RETURN                                                            003560
 500  PRINT *," ERROR IN POLISH NSTK=",NSTK," NPOL=",NPOL               003570
      GO TO 100                                                         003580
      END                                                               003590
      SUBROUTINE PUTNAME(IN,LEN,TYPE,NMPOS)                             003600
      IMPLICIT INTEGER(A-Z)                                             003610
C      PUTNAME STORES A STRING OF CHARACTERS IN THE NAME ARRAY          003612
C        AND PASSES BACK TO SUPERORDINATE A POINTER TO THE              003613
C          POSITION IN WHICH THE STRING WAS STORED IN THE NAME ARRAY.   003614
      COMMON/NAMES/ NAME(100,7),LSTN                                    003620
      DIMENSION IN(1)                                                   003630
      NMPOS=LSTN+1                                                      003650
C        COMPUTE # OF WORDS REQUIRED  TO STORE THE STRING               003654
      NWORDS=(LEN+1)/4 +1                                               003660
C          STORE LENGTH OF STRING BESIDE THE 1ST WORD OF THE STRING     003665
      NAME(LSTN+1,2)=LEN                                                003670
      NAME(LSTN+1,3)=TYPE                                               003680
C          STORE REST OF STRING                                         003684
      DO 5 J=1,NWORDS                                                   003690
      LSTN=LSTN+1                                                       003700
```

```
      DATA EOS/3REOS/,CONT/1H*/                                          004240
      FRST=1                                                             004250
      LST=72                                                             004260
    3 PRINT (1,5) (OUT(I),I=FRST,LST)                                    004270
    5 FORMAT(72R1,8X)                                                    004280
      IF(OUT(LST+1).EQ.EOS)    GO TO 30                                  004290
C       EOS EXPECTED IN COLUMN 73 OR 145 OR 217,ETC                      004294
C   ***PUT IN CONTINUATION MARK                                          004300
      OUT(LST+5)=CONT                                                    004310
      LST=LST+72                                                         004320
      FRST=FRST+72                                                       004330
      GO TO 3                                                            004340
   30 RETURN                                                             004350
      END                                                                004360
      SUBROUTINE GETSTMT(STMT),RETURNS(N1)                               004370
C       ALT RETURN N1 IS USED WHEN AN EOF HAS BEEN DETECTED              004380
C        GETSTMT READS INPUT PROGRAMS CARDS,PASSING A ' STATEMENT'       004382
C          AT A TIME TO ITS SUBORDINATE(GETOBJ).  A STATEMENT            004384
C       MAY SPAN SEVERAL CARDS THROUGH THE USE OF CONTINUATION CARDS,    004385
C          SO A LOOK-AHEAD READ IS REQUIRED.                             004386
      IMPLICIT INTEGER (A-Z)                                             004390
      DIMENSION STMT(1)                                                  004400
      COMMON/CNTS/LNCNT                                                  004420
      COMMON/TEMPRO/TEMP(73)                                             004430
      COMMON/NAMES/NAME(100,3),LSTN                                      004440
      DATA LSTCOL/72/,EOS/3REOS/,ZERO/1H0/,BLNK/1H /,ENDFILE/3REOF/,     004450
     *   MAXCOLS/144/,LNCNT/1/                                           004460
C     USING NO INITIAL READ , JUST A WORD OF BLANKS                      004470
    1 LAST=LSTCOL                                                        004480
C     INITIALIZE LSTN FOR NAMES                                          004490
      LSTN=0                                                             004500
C       CHECK FOR COMMENT CARD                                           004502
      IF(TEMP(1).EQ.1HC.OR.TEMP(1).EQ.1H*.OR.TEMP(1).EQ.1HI) GO TO 70    004510
      DO 2 I=1,LSTCOL                                                    004520
    2 STMT(I)=TEMP(I)                                                    004530
    4 CONTINUE                                                           004550
C     READ INPUT PROGRAM CARD(COLS 1-72) INTO TEMP                       004553
      READ(2,3)  (TEMP(I),I=1,LSTCOL)                                    004560
    3 FORMAT(72A1)                                                       004570
      IF(EOF(2).NE.0)       GO TO 50                                     004580
      LNCNT=LNCNT+1                                                      004590
C     CHECK FOR CONTINUE CARDS--------                                   004600
      IF(TEMP(1).EQ.1H0.OR.TEMP(1).EQ.1H+.OR.TEMP(1).EQ.1HI)   GO TO 30  004610
C     IF NOT A CONTINUED LINE GO TO 30                                   004620
      IF(TEMP(6).EQ.ZERO.OR.TEMP(6).EQ.BLNK)   GO TO 30                  004630
C      GOT A CONTINUATION LINE                                           004640
C       MAXCOLS FOR PRESENT PREPROCESSOR IS 144(2 CARDS)                 004650
C        ALL STATEMENTS LONGER THAN THIS WILL NOT BE ANALYZED            004652
      IF(LAST.GE.MAXCOLS)     GO TO 60                                   004660
      TEMP(6)=BLNK                                                       004670
      DO 14 I=1,LSTCOL                                                   004680
      LAST=LAST+1                                                        004690
   14 STMT(LAST)=TEMP(I)                                                 004700
      GO TO 4                                                            004710
C      PUT EOS MARKER IN LAST COLUMN+1 OF OUTGOING STATEMENT             004713
   30 STMT(LAST+1)=EOS                                                   004720
      RETURN                                                             004730
C      EOF HAS BEEN FOUND (RETURN N1)                                    004740
   50 STMT(LAST+1)=EOS                                                   004750
      STMT(LAST+2)=ENDFILE                                               004760
      RETURN N1                                                          004770
C       GOT TOO LONG OF A STATEMENT-DO NOT PROCESS IT                    004780
C          (OUTPUT STATEMENT THEN READ NEXT CARD UNTIL A NEW             004782
C          STATEMENT HAS BEEN DETECTED(NO MORE CONTINUES))               004783
   60 STMT(LAST+1)=EOS                                                   004790
      CALL PRNT(STMT)                                                    004800
```

```
      NAME(LSTN,1)=IN(J)                                                003710
    5 CONTINUE                                                          003720
      RETURN                                                            003730
      END                                                               003740
      SUBROUTINE INITPD,RETURNS(N1)                                     003750
      IMPLICIT INTEGER (A-Z)                                            003760
C     INITPD DOES THE INITIAL READ OF THE FORTRAN ALGORITHM             003764
C        IT READS THE CONTENTS OF THE FIRST PROGRAM CARD INTO           003765
C         A TEMPORARY ARRAY WHICH IS USED BY THE GET-STATEMENT          003765
C         MODULE, WHICH FUNCTIONS ON A LOOK-AHEAD BASIS.                003767
      COMMON/TEMPPD/TEMP(73)                                            003770
      DATA TEMP/72*1H ,3REOS/                                           003780
      READ(2,23) (TEMP(I),I=1,72)                                       003790
   23 FORMAT(72R1)                                                      003800
C       CHECK FOR EMPTY FILE                                            003802
      IF(EOF(2).NE.3)   RETURN N1                                       003810
      RETURN                                                            003830
      END                                                               003840
      SUBROUTINE GETOBJ(STMT,F),RETURNS(N1)                             003850
C      ALT RETURN IS USED ONCE AN EOF HAS BEEN DETECTED                 003860
C     GETOBJ SUPERVISES THE CHECK OF A STATMENT FOR A LEGAL OBJECT      003865
C        WHERE AN ARITHMETIC STATEMENT STRUCTURE CONSISTS OF            003867
C          OBJECT = ARITHMETIC EXPRESSION                               003868
      IMPLICIT INTEGER(A-Z)                                             00[illegible]
      DIMENSION STMT(1),IFSTMT(3)                                       003880
      COMMON/POLISH/NPOL,NSTK                                           003890
      DATA IFSTMT/1HI,1HF,1H(/,BLNK/1H /,REGSTMT/6/,IFTYPE/1H(/,        003900
     *     EOF/3REOS/,LPAREN/1H(/,RPAREN/1H)/                           003910
C     INITIALIZE NPOL,NSTK - PTRS FOR THE POLISH STRING,SINCE           003930
C      WE WILL BE BEGINNING A NEW ONE FOR EACH STATEMENT                003933
    5 NPOL=NSTK=1                                                       003940
      CALL GETSTMT(STMT),RETURNS(505)                                   003950
      P=I=BEGSTMT                                                       003960
C     CHECK FOR AN IF STMT TYPE,WHICH MAY CONTAIN AN ARITHMETIC         003980
C      STATEMENT TO BE EXECUTED UPON A TRUE CONDITION                   003982
      DO 15 I=1,3                                                       003990
      CALL CLASIFY(STMT,J),RETURNS(500,500)                             004000
      IF(STMT(J).NE.IFSTMT(I))     GO TO 21                             004010
   15 CONTINUE                                                          004020
C     GOT AN IF STATEMENT, PLACE AN INDICATOR IN POLISH STRING          004030
      PARCNT=1                                                          004040
C      SPACE OVER PREDICATE PORTION OF IF STATEMENT TO GET TO OBJECT    004043
   20 I=I+1                                                             004050
      IF(STMT(J).EQ.RPAREN)       PARCNT=PARCNT-1                       004060
      IF(STMT(J).EQ.LPAREN)       PARCNT=PARCNT+1                       004070
      IF(STMT(J).EQ.EOS)       GO TO 500                                004080
      IF(PARCNT.NE.0)      GO TO 20                                     004090
C       GO BACK UNTIL MATCHING RIGHT PAREN OF IF STMT PREDICATE         004095
C         IS FOUND                                                      004095
      CALL STPOL(IFTYPE)                                                004100
      P=I                                                               004110
   21 CALL ENTOBJ(STMT,P),RETURNS(5)                                    004120
      RETURN                                                            004140
C       EOS FOUND ON ARITHMETIC STATEMENT                               004143
  500 CALL PRNT(STMT)                                                   004150
      GO TO 5                                                           004170
C       FOR CONDITION-POINT LAST STATEMENT(ASSUMED TO BE                004175
C         NON-ARITHMETIC(E.G."END"))                                    004177
  505 CALL PRNT(STMT)                                                   004180
      RETURN N1                                                         004190
      END                                                               004200
      SUBROUTINE PRNT(OUT)                                              004210
      IMPLICIT INTEGER(A-Z)                                             004220
C       PRINT ARRAY OUT STATEMENT, WHICH MAY SPAN SEVERAL CARDS, TO     004227
C      THE NEXT RECORD POSITION ON OUTPUT FILE.                         004224
      DIMENSION OUT(1)                                                  004230
```

```
 61   CALL PRNT(TEMP)                                                    004810
C      PUT OUT ALL FURTHER CONTINUATIONS FOR THAT STATEMENT              004820
      LAST=LSTCOL                                                        004830
      READ(1,8) (TEMP(I),I=1,LSTCOL)                                     004840
      IF(EOF(2).NE.0)      GO TO 50                                      004850
      LNCNT=LNCNT+1                                                      004870
C        CHECK FOR COMMENT STATEMENT                                     004875
      IF(TEMP(1).EQ.1RC.OR.TEMP(1).EQ.1R*.OR.TEMP(1).EQ.1R$)   GO TO 69  004880
C      CHECK FOR A CONTINUE STATEMENT                                    004894
      IF(TEMP(5).EQ.7HCONTIN.OR.TEMP(6).EQ.BLNK)    GO TO 1              004990
      GO TO 61                                                           004920
 69   CALL PRNT(TEMP)                                                    004930
      GO TO 61                                                           004940
C     GOT A COMMENT---PRINT IT AND READ NEXT CARD                       004950
 70   CALL PRNT(TEMP)                                                    004960
      READ(2,8)       (TEMP(I),I=1,LSTCOL)                               004970
      IF(EOF(2).NE.0)      GO  TO 50                                     004980
      LNCNT=LNCNT+1                                                      004990
      GO TO 1                                                            005000
      END                                                                005010
      SUBROUTINE ENDOBJ(STMT,P),RETURNS(N1)                              005020
C                                                                        005023
C        ENDOBJ DETERMINES IF A STATEMENT HAS A LEGAL OBJECT,            005030
C     ALT RETURN N1 IS USED WHEN AN ILLEGAL ARITHETIC STMT               005031
C                                                                        005036
      IMPLICIT INTEGER (A-Z)                                             005040
      DIMENSION STMT(1)                                                  005050
      DATA EQ_TYPE/4/                                                    005060
      CALL CLASIFY(STMT,P,TYPE),RETURNS(200,200)                         005070
C      CHECK OBJECT OF ARITHEITIC STATEMENT TO SEE IF IT'S A LEGA-       005075
C         OPERAND/OR ARRAY                                               005077
      CALL OPRND(STMT,P,TYPE),RETURNS(200,201,200)                       005080
      IF(TYPE.NE.EQLTYPE)    GO TO 200                                   005090
C     STORE EQUAL SIN IN POLISH STRING                                   005100
C     NOW GO PROCESS REST OF ARITHMETIC STATEMENT                        005105
      CALL STPOL(STMT(P))                                                005110
      RETURN                                                             005120
C     GOT NON-ARITHMETIC STATEMENT- GO CHECK FOR KEY STATEMENT TYPES     005123
 200  CALL NONEXP(STMT)                                                  005130
      RETURN N1                                                          005140
      END                                                                005150
      SUBROUTINE NONEXP(STMT)                                            005160
      IMPLICIT INTEGER(A-Z)                                              005170
C                                                                        005171
C        NONEXP CALL FOR THE PRINT OF A NON-EXPRESSION STATEMENT         005172
C          AND DETERMINES IF THE NON-EXPRESSION IS A KEY STMT            005173
C          TYPE,FOR DIMENSIONING ARRAYS OR BEGINNING NEW ROUTINES        005174
C                                                                        005179
      DIMENSION KYWRDS(18,6),STMT(1)                                     005180
      DATA KYWRDS/8,1HD,1HI,1HM,1HC,1HN,1HS,1HI,1HO,1HN,3,3,0,0,0,3,     005190
     *      18,1HS,1HU,1HB,1HR,1HO,1HU,1HT,1HI,1HN,1HE,3,3,3,3,3,       005200
     *       9,1HF,1HU,1HN,1HC,1HT,1HI,1HO,1HN,3,3,3,3,3,3,3,            005210
     *       7,1HB,1HE,1HG,1HN,1HE,1HA,1HN,3,3,3,3,3,3,3,3,             005220
     *      12,1HE,1HE,1HA,1HL,1HE,1HU,1HN,1HC,1HT,1HI,1HO,1HN,3,      005230
     *                ,3,                                               005240
     *      17,1HI,1HN,1HT,1HE,1HG,1HE,1HR,1HR,1HU,1HN,1HC,1HT,        005250
     *            1HT,1HO,1HN/                                         005260
      DATA  BLNK/1H /                                                    005270
C     CHECK FOR KEYWORDS DIMENSION,SUBROUTINE,PROGRAM,REAL FUNCTION,     005290
C        INTEGER FUNCTION , AND FUNCTION                                 005296
      I=3                                                                005300
      L2=0                                                               005310
      DO  60 I=1,3                                                       005320
      L=0                                                                005330
      LEN=KYWRDS(I,1)+1                                                  005340
      DO 55 J=2,LEN                                                      005350
```

```
C       MTCH INPUT STATEMENT AGAINST KEYWORDS(BLANKS ARE IGNORED)        005354
 17     P=P+1                                                            005360
        IF(STMT(P).EQ.BLNK)  GO TO 17                                    005370
        IF(STMT(P).NE.KYWRDS(J,I))   GO TO 20                            005380
 15     CONTINUE                                                         005390
C     I=1 INDICATES IT WAS A DIMENSION STATEMENT                         005395
        IF(I.EQ.1)   GO TO 300                                           005400
        GO TO 400                                                        005410
 20     CONTINUE                                                         005420
C IT DID NOT MATCH ANY KEYWORDS.                                        005430
        GO TO 530                                                        005440
 300    CALL DIMENSN(STMT,P)                                             005450
        GO TO 530                                                        005460
C    GOT A NEW ROUTINE-SO SAVE NAME AND INITIALIZE VIA NWRUTIN           005470
 400    CALL NWRUTIN(STMT,P)                                             005480
 530    CALL PRNT(STMT)                                                  005490
        RETURN                                                           005500
        END                                                              005510
        SUBROUTINE NWRUTIN(STMT,P)                                       005520
        IMPLICIT INTEGER(A-Z)                                            005530
C                                                                        005531
C       NWRUTIN EXTRACTS THE PROGRAM UNITS NAME AND INITIALIZES THE      005532
C       LINE COUNTER FOR THAT NEW UNIT TO ONE.(THIS INFORMATION IF       005533
C        IS USED FOR EACH ARITHMETIC FUNCTION SUBROUTINE CALL BUILT      005534
C        BY THE EFFERENT BRANCH).  ALSO IT CLEARS THE LIST OF VARIABLE   005535
C        NAMES DECLARED AS ARRAYS FOR THE PREVIOUS PROGRAM UNIT.         005536
C                                                                        005537
C--- SAVES ROUTINE NAME AND REINITIALZIES LINE COUNT                     005540
C                                                                        005549
        DIMENSION STMT(1)                                                005550
        COMMON/ARRAYS/ ARRAY(40),ARRYMAX                                 005560
        COMMON/GOTON/ GOTONUM                                            005570
        COMMON/ROUTNAM/ ROUTINE                                          005580
        COMMON/CNTS/ LNCNT                                               005590
        DATA LPAREN/12(/,EOS/3REOS/,BLANKS/10H          /,BLNK/1H /      005600
        ROUTINE=BLANKS                                                   005610
        RCNT=0                                                           005620
C     INITIALIZE GOTO # FOR THE NEW ROUTINE JUST DETECTED                005623
C       INITIALIZE LINE COUNT TO 1 FOR NEW ROUTINE                       005624
C       CLEAR ARRAY COUNT(SINCE THIS NEW ROUTINE DOESN'T HAVE            005625
C          ANY AS YET)                                                   005626
        GOTONUM=10                                                       005630
        LNCNT=1                                                          005640
        ARRYMAX=0                                                        005650
 5      P=P+1                                                            005660
C       LEFT PAREN INDICATES END OF ROUTINE NAME                         005665
        IF(STMT(P).EQ.LPAREN)  GO TO 80                                  005670
        IF(STMT(P).EQ.BLNK)  GO TO 5                                     005680
        IF(STMT(P).EQ.EOS)   GO TO 80                                    005690
C       STORE CHARACTER INTO OUTINE NAME SAVE WORD                       005695
        LEFT=(3-RCNT)*6                                                  005700
        ROUTINE=SHIFT(STMT(P),LEFT).OR.(SHIFT(MASK(54),LEFT).AND.ROUTINE) 005710
        RCNT=RCNT+1                                                      005720
        GO TO 5                                                          005730
 80     RETURN                                                           005740
        END                                                              005750
        SUBROUTINE DIMENSN(STMT,P)                                       005760
C                                                                        005769
C       DIMENSN STORES ALL VARIABLE NAMES WHICH HAVE BEEN DECLARED       005775
C         ARRAYS BY THE DIMENSION STATEMENT                              005776
C                                                                        005777
C  POINTER P IS EXPECTED TO BE SETTING AT LAST LETTER OF DIMENSION WORD  005779
        IMPLICIT INTEGER (A-Z)                                           005780
        COMMON/ARRAYS/ARRAY (40),ARRYMAX                                 005790
        DIMENSION STMT(1)                                                005800
        DATA LPAREN/1H(/,RPAREN/1H)/,EOS/3REOS/,BLNK/1H /,               005810
```

```
      *  COMMA/1R,/,BLANKS/10H          /
 4       NAM=BLANKS
         NCNT=0
 5       P=P+1
C         CHECK FOR BEGINNING OF DIMENSION
         IF(STMT(P).EQ.LPAREN)    GO TO 60
         IF(STMT(P).EQ.BLNK)   GO TO 5
         IF(STMT(P).EQ.EOS)   GO TO 80
         LEFT=(9-NCNT)*6
C         STORE AWAY DIMENSIONED VARIABLE NAME(ONE CHARACTER AT A TIME)
         NAM=SHIFT(STMT(P),LEFT).OR.(MASK(NCNT*6).AND.NAM)
         NCNT=NCNT+1
         GO TO 5
C          HAS BEEN DETECTED
C WE HAVE FOUND A NEW ARRAY SO SAVE IT THEN PROCESS ITS ARGUMENTS
 60      PCNT=1
         ARRYMAX=ARRYMAX+1
         ARRAY(ARRYMAX)=NAM
C     PROCESSING UNTIL WE FIND RIGHT PAREN
 61      P=P+1
         IF(STMT(P).EQ.RPAREN)    GO TO 67
C     IF EOS -PROBABLE ERROR STOP PROCESSSING
         IF(STMT(P).EQ.EOS)   GO TO 80
         IF(STMT(P).EQ.LPAREN)    PCNT=PCNT+1
         GO TO 61
 67      IF(PCNT.EQ.1)    GO TO 74
         PCNT=PCNT-1
         GO TO 61
C COMMA SEPARATES ARRY DECLARATIONS
 74      P=P+1
         IF(STMT(P).EQ.COMMA)    GO TO 4
         IF(STMT(P).EQ.EOS)   GO TO 80
         GO TO 74
 80      RETURN
         END
         SUBROUTINE SETUP
         IMPLICIT INTEGER (A-Z)
C        SETUP SCANS POLISH STRING LEFT TO RIGHT AND CALLS A HANDLING
C           ROUTINE FOR EACH OPERATOR FOUND.
         DIMENSION NAME(3),TARRAY(6),OBJECT(6)
         DIMENSION OPS(9),FLLWER(3)
         DIMENSION CONT(2)
         COMMON/GOTON/GOTONUM
         COMMON/CNTS/LNCNT
         COMMON/ROUTNA/ ROUTINE
         COMMON/POL/POLISH(80,2) T
         DATA ITEMPN1/7HITTTTAA/,RTEMPN2/7HRTTTTAA/,EOL/-44/,NEG/1H-/
C     ORDER OF OPS IS + - * / ** = , ? !
         DATA OPS/-37,-38,-39,-40,-43,-44,-46,-57,-63/
         DATA IFTYP/1R3/,GOTONUM/10/,CONT/10H       CONT, 10HINUE        /,
      *     CONTLEN/14/
         ITEMPN=ITEMPN1
         RTEMPN=RTEMPN2
C   .   FNL IS A FLAG WHICH IDICATES THE LAST OPERATION PERFORMED
C           IN THE ARITHMETIC STATEMENT (DESCRIBED BY THE POLISH NOTATION)
C
         FNL=0
         FLLWLEN=21
C          BUILD STANDARD FOLLOWER FOR ALL SUB CALLS DERIVED FROM THE
C           POLISH STRING OF A SINGLE ARITHMETIC STATEMENT
         ENCODE(FLLWLEN,1,FLLWER)   ROUTINE,LNCNT,FNL
 1       FORMAT(3H,7A,A7,1H.,I3,1H,,I1,9H,KEY))
 2       P=1
C   -----OPERATORS ARE CODED TO BE LESS THAN 0 IN POLISH STRING
 5       IF(POLISH(P,1). LT.0) GO TO 15
         IF (POLISH (P,1). EQ. ZEROS) GO TO 500
```

```
      P=P+1                                                              006410
      GO TO 5                                                            006420
C        FIND NEXT P                                                     006430
15    NP=P+1                                                             006440
17    IF (POLISH (NP,1). LT. 0) GO TO 20                                 006450
C   ----- CHECK FOR "=" (SINGLE ASSIGNMENT INDICATION)                   006460
      IF (POLISH (NP,1). EQ. 3REOS) GO TO 40                             006470
      NP=NP+1                                                            006480
      GO TO 17                                                           006490
20    IF (POLISH (NP,1). EQ. EOL) ENL=1                                  006500
      ENCODE(FLLW_EN,1,FLLWER) ROUTINE,LNCNT,ENL                         006510
      GO TO 70                                                           006520
C     *******  "=" SHOULD ALWAYS SET ENL=1 AND STOP IT AL_               006530
C   ---SHOULD BE AN EQUAL SIGN IF IT IS THE ONLY OPERATOR IN STRING      006540
40    IF (POLISH (P,1). EQ. EOL) GO TO 42                                006550
      PRINT*, "ERROR-NO EQUAL"                                           006560
      RETURN                                                             006570
C        SET UP A SINGLE ASSIGNMENT                                      006580
 42   CALL GETRND(POLISH,P,RND)                                          006590
      CALL GETNAME(POLISH(RND,1),NAME,TYPE,LEN)                          006600
C   NOW STORE THE OBJECT                                                 006610
      CALL GETRND(POLISH,RND,OBJ)                                        006620
      CALL GETNAME(POLISH(OBJ,1),OBJECT,OBTYPE,OBLEN)                    006630
C     -- NO NEED TO STORE NAME SINCE IT IS THE LAST ONE                  006640
      CALL SNGLASS(NAME,LEN,TYPE,OBJECT,OBLEN,FLLWER,FLLW_EN)            006650
C     --- NOW CLEAR POLISH                                               006660
      POLISH(OBJ,1)=)                                                    006670
      POLISH(RND,1)=)                                                    006680
C     ---  GO GET NEXT POLISH STRING OPERATOR                            006690
      GO TO 240                                                          006700
 70   DO 72 I=1,3                                                        006710
      IF(POLISH(P,1).EQ.OPS(I))  GO TO 74                                006720
 72   CONTINUE                                                           006730
C    ******** THE FOLLOWING IS TEMPORARY **********                     006740
      PRINT *," BAD OPERATOR"                                            006750
      RETURN                                                             006760
 74   IF(I.LE.3)     GO TO 230                                           006770
      IF(I.EQ.7)    GO TO 117                                           006780
      IF(I.EQ.8)    GO TO 85                                            006790
      IF(I.EQ.9)    GO TO 125                                           006800
      RETURN                                                             006810
C        GOT A FINISH FUNCTION                                           006820
 85   CALL FINSFN(POLISH,P,ITEMPN,RTEMPN,ENL,FLLWER,FLLW_EN)             006830
      GO TO 241                                                          006840
C        GOT "," BUILD PART OF A FUNCTION                                006850
C     *******CALL FUNCBLD(POLISH,P)  **********                         006860
 117  CONTINUE                                                           006870
      CALL FUNCBLD(POLISH,P,ITEMPN,RTEMPN,FLLWER,FLLWLEN)                006880
      GO TO 240                                                          006890
C        GOT UNARY                                                       006900
 125  CALL GETRND(POLISH,P,RND)                                          006910
      CALL GETNAME(POLISH(RND,1),NAME,TYPE,LEN)                          006920
      TARRAY(1)=NEG                                                      006930
      TLEN=1                                                             006940
      CALL MERGE(TARRAY,TLEN,NAME,LEN)                                   006950
      CALL PUTNAME(TARRAY,TLEN,TYPE,PTR)                                 006960
      POLISH(RND,1)=PTR                                                  006970
      GO TO 241                                                          006980
 230  CALL GETRTS(POLISH,P,ITEMPN,RTEMPN,ENL,FLLWER,FLLW_EN)            006990
C      GOT TO GET NEXT OPERATOR                                          007000
 240  IF(ENL.EQ.1)    GO TO 500                                          007010
 241  P=P                                                                007020
      GO TO 5                                                            007030
 500  CONTINUE                                                           007040
C     ********END OF STMT OR IF FOLLOWING**********                     007050
      IF(POLISH(P,1).NE.IFTYP)     GO TO 525                             007060
```

```
      ENCODE(CONTLEN,99,CONT)   GOTONUM                                  007100
   99 FORMAT(3H Z1,I2,9H CONTINUE)                                       007110
      CALL PRINTIT(CONT,CONTLEN)                                         007120
  525 CONTINUE                                                           007130
      RETURN                                                             007140
C     LABEL 500 MAY WANT TO SET A FLAG OR ALTERNATE RETURN *****         007160
      END                                                                007170
      SUBROUTINE FINSENC(POLISH,P,ITEMPN,RTEMPN,FNL,FLLWER,FLLWLEN)      007180
      IMPLICIT INTEGER(A-Z)                                              007190
C                                                                        007192
C        FINSENC MERGES THE FUNCTION NAME WITH ITS ARGUMENT LIST         007193
C        (BUILT BY FUNCBLD) AND GENERATES A SINGLE ASSIGNMENT            007194
C        CALL FOR THE FUNCTION.                                          007195
C                                                                        007199
      DIMENSION POLISH(50,2),FARGS(20),FNAM(20),OBJECT(2)                007200
      DIMENSION FLLWER(1)                                                007210
      DATA LPAREN/1L(/,RPAREN/1L)/,EVAL/1/                               007220
      DATA FARGS/20*10H          /,FNAM/20*10H          /                007230
C     GET FUNCTION ARGUMENTS                                             007240
      CALL GETRND(POLISH,P,RND)                                          007250
      IF(POLISH(RND,2).EQ.EVAL)    GO TO 15                              007260
C     ----   GET A TEMPORARY OBJECT FOR THE SINGLE ARGUMENT  ----        007270
      CALL GETNAME(POLISH(RND,1),FARGS,TYPE,FLEN)                        007280
      OBLEN=7                                                            007290
      IF(TYPE.EQ.2)   GO TO 7                                            007300
      OBJECT(1)=ITEMPN                                                   007310
      ITEMPN=ITEMPN+1000000B                                             007320
      GO TO 11                                                           007330
    7 OBJECT(1)=ITEMPN                                                   007340
      ITEMPN=ITEMPN+1000000B                                             007350
   11 CALL SNGLARG(FARGS,FLEN,TYPE,OBJECT,OBLEN,FLLWER,FLLWLEN)          007360
      FARGS(1)=OBJECT(1)                                                 007370
      ARGLEN=OBLEN                                                       007380
      GO TO 17                                                           007390
   15 CALL GETNAME(POLISH(RND,1),FARGS,DUMMY,ARGLEN)                     007400
C     NOW GO GET FUNCTION NAME                                           007410
   17 CALL GETRND(POLISH,RND,FNC)                                        007420
      CALL GETNAME(POLISH(FNC,1),FNAM,TYPE,FLEN)                         007430
      CALL MERGE(FNAM,FLEN,LPAREN,1)                                     007440
      CALL MERGE(FNAM,FLEN,FARGS,ARGLEN)                                 007450
      CALL MERGE(FNAM,FLEN,RPAREN,1)                                     007460
      POLISH(RND,1)=,                                                    007470
      IF(FNL.EQ.1)     GO TO 50                                          007480
      OBLEN=7                                                            007490
      IF(TYPE.EQ.2)    GO TO 35                                          007500
      OBJECT(1)=ITEMPN                                                   007510
      ITEMPN=ITEMPN+1000000B                                             007520
      GO TO 40                                                           007530
   35 OBJECT(1)=ITEMPN                                                   007540
      ITEMPN=ITEMPN+1000000B                                             007550
   40 CONTINUE                                                           007560
      OBJTYPE=TYPE                                                       007570
      CALL SNGLARG(FNAM,FLEN,OBJTYPE,OBJECT,OBLEN,FLLWER,FLLWLEN)        007580
C  ***** BELOW IS STORING AWAY PTR TO THE WHOLE FUNCTION*****            007590
C     --- WAS THAT THE LAST ASSIGNMENT (FNL=1)??   -- IF SO RETURN      007600
      CALL PUTNAME(OBJECT,OBLEN,TYPE,PTR)                                007610
      POLISH(FNC,1)=PTR                                                  007620
      GO TO 60                                                           007630
   50 CALL GETRND(POLISH,FNC,LAST)                                       007640
      CALL GETNAME(POLISH(LAST,2),OBJECT,DUMMY,OBJLEN)                   007650
      POLISH(LAST,2)=EVAL                                                007660
      CALL SNGLARG(FNAM,FLEN,TYPE,OBJECT,OBJLEN,FLLWER,FLLWLEN)          007670
   60 RETURN                                                             007680
      END                                                                007690
      SUBROUTINE SETLIST(LOCS, OUT, TYPE, LOC)                           007700
      IMPLICIT INTEGER (A-Z)                                             007710
```

```
C                                                                       007711
C     GETNAME GETS A STRING OF CHARACTERS FROM THE NAME ARRAY           007713
C      ACCORDING TO THE POINTER GIVEN AND PASSES TO THE CALLER          007714
C        THE STRING, ITS DATA TYPE, AND ITS LENGTH                      007715
C                                                                       007716
      COMMON/NAMES/ NAME(100,3),LSTN                                    007720
      DIMENSION OUT(1)                                                  007730
      LEN=NAME (NPOS,2)                                                 007740
      TYPE=NAME (NPOS,3)                                                007750
      NWRDS= (LEN-1)/10+1                                               007760
      DO 5 J=1, NWRDS                                                   007770
      OUT(J) = NAME (NPOS,1)                                            007780
5     NPOS=NPOS+1                                                       007790
      RETURN                                                            007800
      END                                                               007810
      SUBROUTINE PRINTIT (OUT, LEN)                                     007820
C                                                                       007823
C         PRINTS 80 CHARACTER CARD IMAGES TO THE N-BIT SIMULATED        007824
C           PROGRAM OUTPUT FILE                                         007825
C       TAKES CARE OF CONTINUATIONS                                     007830
      IMPLICIT INTEGER (A-Z)                                            007840
      DIMENSION OUT(1)                                                  007850
      DATA CONTINU/1HX     *    /                                       007860
      NWRDS=(LEN-1)/10+1                                                007870
      J=1                                                               007880
      K=7                                                               007890
      IF(NWRDS.LT.7)    K=NWRDS                                         007900
C      PRINT CARD IMAGE TO OUTPUT FILE                                  007910
      PRINT(1,5) (OUT(I),I=J,K)                                         007920
5     FORMAT (7A10)                                                     007930
      IF(NWRDS.LE.7)     GO TO 20                                       007940
      K=K+6                                                             007950
C       PRINT SUCCEEDING CARD IMAGES AS CONTINUED STATEMENT             007954
      DO 14 J=8, NWRDS, 6                                               007960
      IF(K.GT.NWRDS)    K=NWRDS                                         007970
      PRINT (1,5) CONTINU, (OUT(I), I=J,K)                              007980
      JJ=J                                                              007990
      K=K+6                                                             008000
14    CONTINUE                                                          008010
      K=K-6                                                             008020
C       CLEAR OUT 81 IT WON'T BE PRINTED ON SUCCEEDING PRINTS           008021
C         OF CREATED MORE THAN ONE CARD                                 008022
21    DO 23 LL=JJ,K                                                     008030
23    OUT(LL)=1HX                                                       008040
      RETURN                                                            008050
      END                                                               008060
      SUBROUTINE FUNCBLD(POLISH,P,TEMPNM1,TEMPNM2,FLLWEP,FLLWLEN)       008070
      IMPLICIT INTEGER (A-Z)                                            008080
C                                                                       008081
C       FUNCBLD MERGES FUNCTION ARGUMENTS TOGETHER,GENERATING           008082
C         CALLS FOR SINGLE ARGUMENTS IF NECESSARY.                      008083
      DIMENSION FLLWEP(1)                                               008090
      DIMENSION POLISH(50,1),NAME(20),NAME2(20)                         008100
      DATA EVAL/1/, COMMA/1H,/                                          008110
      DATA NAME/20*1H           /,NAME2/20*1H           /               008120
C      TLEN1 AND TLEN2 REPRESENT LENGTH OF TEMPORARY NAME               008123
      TLEN1=TLEN2=7                                                     008130
      CALL GETARG(POLISH, P, ARG)                                       008140
C     IF ITS BEEN EVALUATED- GO ON,OTHERWISE CREATE A SINGLE            008150
C       INSTRUCTION CALL FOR THE ARGUMENT (SO THAT IS TAKES             008152
C       ON THE N-BIT SIGNIFICANCE).                                     008153
C                                                                       008154
      IF (POLISH(P+1,2). EQ. EVAL) GO TO 40                             008160
C       GET FUNCTION ARGUMENT                                           008167
      CALL GETNAME (POLISH(ARG,1), NAME, TYPE, LEN)                     008170
      IF(TYPE.EQ.3)   GO TO 32                                          008200
```

```
      TEMPN=TEMPNM2                                                     008210
C       INCREMENT TO NEXT REAL TEMPORARY VARIABLE NAME                  008212
      TEMPNM2=TEMPNM2+10000000                                          008220
      GO TO 29                                                          008230
C     INCREMENT TO NEXT TEMPORARY INTEGER VARIABLE NAME                 008231
 22   TEMPN=TEMPNM1                                                     008240
      TEMPNM1=TEMPNM1+10000000                                          008250
 29   CONTINUE                                                          008260
C       SET UP CALL FOR STORING AWAY THE TEMPORARY OBJECT NAME          008263
      CALL SNGLASG(NAME, LEN, TYPE, TEMPN, TLEN2,FLLWER,FLLWLEN)        008270
      CALL PUTNAME(TEMPN,TLEN2,TYPE,PTR)                                008280
      POLISH (RND,1)=PTR                                                008290
      POLISH (RND,2)=EVAL                                               008300
C       GET OTHER FUNCTION ARGUMENT                                     008302
 40   CALL GETRND (POLISH, RND, RND1)                                   008310
C   SEE IF IT HAS ALREADY BEEN EVALUATED                                008320
      IF (POLISH (RND1,2). EQ. EVAL) GO TO 80                           008330
C     IT HASN'T BEEN EVALUATED,SO EVALUATE IT WITH A SUB CALL           008334
      CALL GETNAME (POLISH (RND1,1), NAME, TYPE, LEN)                   008340
C      CREATE TEMPORARY VARIABLE NAMES                                  008342
      IF(TYPE.EQ.2)   GO TO 45                                          008350
      TEMPN=TEMPNM2                                                     008360
      TEMPNM2=TEMPNM2+10000000                                          008370
      GO TO 49                                                          008380
 45   TEMPN=TEMPNM1                                                     008390
      TEMPNM1=TEMPNM1+10000000                                          008400
 49   CONTINUE                                                          008410
C       EVALUATE ARGUMENT WITH SINGLE ASSIGNMENT CALL                   008414
      CALL SNGLASG(NAME,LEN,TYPE,TEMPN,TLEN1,FLLWER,FLLWLEN)            008420
      CALL PUTNAME(TEMPN,TLEN1,TYPE,PTR)                                008430
      POLISH (RND1,1)=PTR                                               008440
      POLISH (RND1,2)=EVAL                                              008450
C       NOW TAKE EVALUATED ARGUMENTS AND MERGE THEM TOGETHER            008453
C       TO BUILD PART OF THE FUNCTION ARGUMENT LIST                     008455
 80   CALL GETNAME(POLISH(RND1,1),NAME,TYPE,LEN)                        008460
      CALL GETNAME (POLISH (RND,1), NAME2, TYPE, LEN2)                  008470
      CALL MERGE(NAME,LEN,COMMA,1)                                      008480
      CALL MERGE(NAME,LEN,NAME2,LEN2)                                   008490
      CALL PUTNAME(NAME,LEN,TYPE,PTR)                                   008500
C   ARG1 HAS ALREADY BEEN MARKED AS EVALUATED AND ZERO OUT ARG 2        008510
      POLISH(RND1,1)=PTR                                                008520
      POLISH(RND,1)=)                                                   008530
      RETURN                                                            008540
      END                                                               008550
      SUBROUTINE SNGLASG(OPRND,OPRLEN,TYPE,OBJECT,OBJLEN,FLLWER,FLLWLEN)008560
      IMPLICIT INTEGER(A-Z)                                             008570
C                                                                       008573
C         SNGLASN MERGES PARTS REQUIRED FOR A SINGLE ASSIGNMENT         008575
C          FUNCTION CALL                                                008576
C                                                                       008577
      DIMENSION OPRND(1),OBJECT(1),OUT(50),FLLWER(1)                    008580
      DATA ASSN/7H=ASSN(/,IASSN/7H=IASSN(/,LOP/64      /                008590
      DATA OUT/66*1H      /                                             008600
      OUT(1)=LOP                                                        008610
      OLEN=6                                                            008620
C PUT IN OBJECT                                                         008630
      CALL MERGE(OUT,OLEN,OBJECT,OBJLEN)                                008640
      SUB=ASSN                                                          008650
      IF(TYPE.EQ.2)   SUB=IASSN                                         008660
      SUBLEN=7                                                          008670
      CALL MERGE(OUT,OLEN,SUB,SUBLEN)                                   008680
      CALL MERGE(OUT,OLEN,OPRND,OPRLEN)                                 008690
      CALL CHECKCH(OLEN,OUT)                                            008700
      CALL MERGE(OUT,OLEN,FLLWER,FLLWLEN)                               008710
      CALL PUTNTT(OUT,OLEN)                                             008720
      RETURN                                                            008730
```

```
      END
      SUBROUTINE GETRND(POLISH,LP,NXT)
      IMPLICIT INTEGER(A-Z)
      DIMENSION POLISH(1,1)
C
C     GETRND FETCHS (FROM THE POLISH STRING) THE LAST OPERAND
C       AND PASSES BACK A POINTER TO THAT OPERAND
C
      NXT=LP
 5    NXT=NXT-1
      IF(POLISH(NXT).GT.0)    RETURN
      GO TO 5
      END
      SUBROUTINE MERGE(OUT,OLEN,IN,ILEN)
      IMPLICIT INTEGER(A-Z)
C
C       MERGES STRING ON RIGHT TO THE RIGHT END OF STRING ON THE LEFT
C     OLEN WILL THEN ASSUME THE TOTAL LENGTH OF THE RESULTING STRING
C     AND OUT WILL CONTAIN THE MERGED STRING.
C
      DIMENSION OUT(1),IN(1)
C    WOULD NOT NEED TO INITIALIZE OUT THIS WAY
      OWRDS=(OLEN-1)/10+1
      NIWRDS=(ILEN-1)/10+1
      REM=OLEN-(OLEN/10)*10
      IF(REM.EQ.0)    GO TO 65
C     GOT A PARTIAL WORD- COMPUTE WHAT IS LEFT IN OUT(# CHAR)
      NLTTRS=REM+NIWRDS*10
      ENCODE(NLTTRS,50,OUT(OWRDS)) REM,OUT(OWRDS),NIWRDS,
     *     (IN(I),I=1,NIWRDS)
 50   FORMAT(A=,=(A10))
      GO TO 70
C        TEST FOR OLEN = 0 COVERS CASE WHERE OUT ARRAY WAS EMPTY
 65   IF(OLEN.EQ.0)  OWRDS=0
C        FILL IN WHOLE WORDS
      DO 67 I=1,NIWRDS
      OWRDS=OWRDS+1
 67   OUT(OWRDS)=IN(I)
 70   OLEN=OLEN+ILEN
      RETURN
      END
      SUBROUTINE GETPRTS(POLISH,P,ITEMPN,ITEMPN,FNL,FLLWER,FLLWLEN)
      IMPLICIT INTEGER(A-Z)
C
C      GETPRTS ASSEMBLES AND MERGES PARTS OF A SUBROUTINE CALL
C       INVOLVING TWO OPERANDS.
C
      DIMENSION OPRND1(6),OPRND2(6),OUT(40),FLLWER(1),SUBN(4,5),
     *      POLISH(61,2),OBJECT(2),OPS(2,9)
      DATA BLANKS/10H          /,SUBLEN/7/,
     COMMA/1H,/,TPLEN/7/,NOS/3PROS/,EVAL/1/
      DATA SUBN/7H= RADD(,7H= 2ADD(,7H=RIADD(,7H=IIADD(,
     *    7H=RRMNS(,7H=R2MNS(,7H=RIMNS(,7H=IIMNS(,
     *    7H=RRMPY(,7H=R2MPY(,7H=RIMPY(,7H=IIMPY(,
     *    7H=RRDVD(,7H=R2DVD(,7H=RIDVD(,7H=IIDVD(,
     *    7H=RRTXP(,7H=R2EXP(,7H=RIEXP(,7H=IITXP(/
      DATA OPS/1H+,-37,1H-,-38,1H*,-39,1H/,-40,1H),-43,1H(,-53,
     *    1H=,-44,1H),-37,1H,,-46/
      DATA OUT/40*1H          /,OPRND1/6*10H          /
      DATA OPRND2/6*10H          /
      DO 1  I=1,40
 1    OUT(I)=0
C     MATCH OPERATOR TYPE TO SUBROUTINE TYPE
      DO 15  I=1,9
      IF(OPS(2,I).EQ.POLISH(P))  GO TO 17
 15   CONTINUE
```

```
C     ******??????????????************
        PRINT *," BAD OP P=",P
        RETURN
 17     SN=I
C         GOT OPERAND 2
        CALL GETRND(POLISH,P,RND2)
C       GETNAME  STORES THE NAME INTO OPRND2 ARRAY
 31     CALL GETNAME(POLISH(RND2,1),OPRND2,TYPE2,LEN2)
C       GET OPERAND 1 LOCATION IN POLISH
        CALL GETRND(POLISH,RND2,RND1)
        CALL GETNAME(POLISH(RND1,1),OPRND1,TYPE1,LEN1)
C     MAKE A SSIFT
        MODE=(TYPE2-1)+2*(TYPE1-1)+1
C       FETCH SUBRUTINE NAME
        SUB=SUBN(MODE,SN)
        IF(FNL.EQ.1)     GO TO 70
        IF(MODE.LE.3)    GO TO 60
        OBJECT(1)=ITEMPN
        ITEMPN=ITEMPN+10000008
        TYPE=2
        GO TO 65
C     NEED A REAL OBJECT
 60     OBJECT(1)=RTEMPN
        RTEMPN=RTEMPN+10000003
        TYPE=1
C       STORE AWAY NEW OBJECTS NAME
 65     CALL PUTNAME(OBJECT,TPLEN,TYPE,NPOS)
        POLISH(RND1,1)=NPOS
        POLISH(RND1,2)=EVAL
C       CLEAR OPERAND 2 FROM POLISH STRING ALSO
        POLISH(RND2,1)=0
        POLISH(RND2,2)=0
        OBJLEN=TPLEN
        GO TO 75
C       LAST ONE--GET FINAL OBJECT
 70     CALL GETRND(POLISH,RND1.LAST)
        CALL GETNAME(POLISH(LAST,1),OBJECT,TYPE,OBJLEN)
C       ******JUST IN CASE  SET EVAL IN THIS OBJECT****
        POLISH(LAST,2)=EVAL
 75     OUT(1)=BLANKS
C       FIRST LENGTH WILL BE THE FIRST 6 BLANKS
        OLEN=6
        CALL MERGE(OUT,OLEN,OBJECT,OBJLEN)
        CALL MERGE(OUT,OLEN,SUB,SUBLEN)
        CALL MERGE(OUT,OLEN,OPRND1,LEN1)
        CALL MERGE(OUT,OLEN,COMMA,1)
        CALL MERGE(OUT,OLEN,OPRND2,LEN2)
        CALL CHECKTH(OLEN,OUT)
        CALL MERGE(OUT,OLEN,FLLWER,FLLWLEN)
        CALL PRINTIT(OUT,OLEN)
        RETURN
        END
        SUBROUTINE PRELPRT(STMT)
        IMPLICIT INTEGER(A-Z)
C
C         PRELPRT OUTPUTS A COMMENT STATEMENT CONTAINING THE INPUT
C          STATEMENT AND OUTPUTS PRELIMINARY OUTPUT FOR LABELS AND
C          IF STATEMENTS (WHEN NECESSARY)
C
        DIMENSION STMT(1),OUT(1[illegible]),IFTYPE(3),GOTO(73),CONT(73)
C       CHECK FOR LABELS AND PRINTS THEM
C        WE HAVE GOT AN EXPRESSION AT THIS POINT
C     CHECK FOR IF( ) EXPRESSION STMT
       COMMON/3/[illegible]/ GO TO NUM
       DATA [illegible]/1H /,CONTINU/[illegible]/,[illegible]/[illegible]/,CONT/6*1[illegible] .1[illegible],1R0,1[illegible].1R[illegible].
     *  1R1,1RN,1[illegible],1R[illegible],[illegible] ,[illegible]/,TOS/3*EOS/
```

```
      DATA IFTYPE/1HI,1HF,1H(/,RPAREN/1H)/,LPAREN/1H(/,GOTO/5*1H ,1HG,
     *  1HO, 1H ,1HT,1HO,1H ,1H7,1H9,1H1,1H ,1H ,53*1H ,3HEOS/
      DO 4 I=1,5
      IF(STMT(I).NE.BLNK)    GO TO 30
    4 CONTINUE
C     --- NO LABELS - JUST COMMENT IT
      GO TO 35
   30 DO 33 I=1,5
   33 CMNT(I)=STMT(I)
      CALL PRNT(CMNT)
   35 LST=73
   37 IF(STMT(LST).EQ.EOS)   GO TO 38
      STMT(LST)=CMNT
      LST=LST+72
      GO TO 37
   38 STMT(1)=CMNT
      DO 39 I=2,5
   39 STMT(I)=BLNK
      CALL PRNT(STMT)
C      CHECK FOR AN IF STMT --USE SAME RUTEIN**********
   40 J=6
      DO 43 I=1,3
      CALL CLASIFY(STMT,J,TYPE),RETURNS(500,500)
      IF(STMT(J).NE.IFTYPE(I))    GO TO 500
   43 CONTINUE
C     ----    GOT AN IF STMT    -----
C      NOW PASSOVER IF(=====)---
      PARCNT=1
   45 J=J+1
      IF(STMT(J).EQ.RPAREN)   PARCNT=PARCNT-1
      IF(STMT(J).EQ.LPAREN)   PARCNT=PARCNT+1
      IF(PARCNT.GT.0)  GO TO 45
      J=J+1
      DO 46 I=1,5
   46 OUT(I)=BLNK
      DO 47 K=7,J
   47 OUT(K)=STMT(K)
C     TO COVER IF CONTAINING > 1 STMT (LINE)
      NUM=((J/73)+1)*72
C      -- FILL IN REST OF LINE WITH BLANK, MAKE GO TO POSITION A CONTINUE
      DO 50 I=J,NUM
   50 OUT(I)=BLNK
      OUT(NUM+1)=EOS
      CALL PRNT(OUT)
      GOTONUM=GOTONUM+1
      GOTO(1)=CONTINU
      ENCODE(2,67,GOUT)GOTONUM
   67 FORMAT(I2)
C     GOT TO CONVERT BINARY NUMBER TO ALPHANUMERIC
      DECODE(2,68,GOUT) (GOTO(I),I=15,16)
   68 FORMAT(21,21)
      CALL PRNT(GOTO)
C      --- CREATE THE CONTINUE LINKED WITH THE IF GO TO
      K=6
      DO 70 I=12,16
      K=K+1
   70 OUT(K)=GOTO(I)
      GOTO(1)=BLNK
      GOTONUM=GOTONUM+1
      ENCODE(2,67,OUT)   GOTONUM
      DECODE(2,68,OUT)   (GOTO(I),I=15,16)
      CALL PRNT(OUT)
C     --- CONTINUE FOR EXPRESSION  ----
      CALL PRNT(STMT)
  500 RETURN
      END
```

```
      SUBROUTINE CHECKZH(LEN,OUT)                                       010600
      IMPLICIT INTEGER(A-Z)                                             010610
C                                                                       010612
C     CHECKZH VOERS SPECIAL CASE TO ASSURE "ZH" DOES NOT OVERLAP        010613
C      LINES AND CAUSE A SYNTAX ERROR.  IT FILLS REMAINING PORTION      010614
C      OF THE LINE WITH BLANKS(SHIFTING THE REST DOWN TEN SPACES)       010615
      DIMENSION OUT(1)                                                  010620
      DATA BLANKS/1H          /                                         010630
      IF(LEN.LE.60)    GO TO 99                                         010640
      IF(LEN.GT.69.AND.LEN.LT.70)     GO TO 5                           010650
      IF(MOD(LEN,80).LT.10)     GO TO 5                                 010660
      GO TO 99                                                          010670
 5    CALL MERGE(OUT,LEN,BLANKS,10)                                     010690
 99   RETURN                                                            010690
      END                                                               010700
```

N-bit Simulation Subroutines

Twenty-two n-bit simulation subroutines were built to handle each combination of data types and arithmetic operations. In addition, there were four other subroutines developed (SETNBIT, ROUNDER, OVERFLW, and UNDRFLW).

The SETNBIT subroutine was developed to compute seven key values upon which the other n-bit subroutines base their n-bit wordlength simulation effects. The OVERFLW and UNDRFLW subroutines are used to print overflow messages. A subroutine called ROUNDER is used whenever a special case results where a round up increments the exponent of the floating point value (due to the fixed point add performed to round floating point values).

The 22 n-bit simulation subroutines basically are the same. They can be divided into two classes: those which produce real values and those which produce integer values. After the operation is performed in each subroutine, the rest of the routine is eventually the same as each of the others within its own class.

The source listing of the n-bit simulation subroutines follows:

N-bit Simulation Subroutines

Source Listing

```
      SUBROUTINE SETNBIT(NBITS,IRNDTR,IPRCSN,MESSGS,MANTSA,IEXPNT,      000100
     *     IPTPOS,KEY)                                                  000110
C                                                                       000120
C       THIS ROUTINE ESTABLISHS THE MIN/MAX VALUES THAT COULD BE        000130
C         REPRESENTED IN FIXED AND FLOATING PT AS OF THE GIVEN N-BIT    000140
C           SPECIFICATIONS-ROUTINE TERMINATES PROGRAM FOR ILL           000150
C                                                                       000160
      DIMENSION KEY(7)                                                  000170
      DATA MANTLEN/48/                                                  000180
      IF((MANTSA+IEXPNT+1).NE.NBITS)   GO TO 31                         000190
      IF(NBITS.GT.60)    GO TO 30                                       000200
      IF(IRNDTR.LT.0)     GO TO 32                                      000210
C     ******NOTE THESE CANT USE REGULAR UNITS--DEPENDS ON USER PROGRAM****000220
C     MIGHT WANT INTEGER MAX TO BE NBITS+2 TO MAKE UP FOR SIGN BIT      000230
      KEY(1)=SHIFT(MASK(1),NBITS)-1                                     000240
      IF(IPRCSN.GT.3.OR.IPRCSN.LT.1)     GO TO 33                       000250
      IF(IEXPNT.GT.11)    GO TO 35                                      000260
      IF(MANTSA.EQ.MANTLEN.AND.IRNDTR.NE.0)     GO TO 34                000270
      MAXMANT=MANTSA+(IPRCSN-1)*NBITS                                   000280
      IF(MAXMANT.GT.MANTLEN)      GO TO 34                              000290
      KEY(4)=IRNDTR                                                     000300
C       SET MESSAGE PRINT/SUPPRESSION FLAG                              000310
      KEY(5)=MESSGS                                                     000320
      KEY(6)=MANTLEN-MANTSA                                             000330
      KEY(7)=MANTLEN-MAXMANT                                            000340
C     ******NEED 60 BIT ALLOWANCE IN THIS ONE AS OF 17 NOVEMBER****     000350
      IF(IEXPNT.EQ.11.AND.MANTSA.EQ.48)  GO TO 25                       000360
      IRNGE=SHIFT(MASK(1),IEXPNT)-1                                     000370
      BIG=2.**IRNGE                                                     000380
      IF(IPTPOS.EQ.0)      BIG=2.**(IRNGE-MANTSA)                       000390
      IPMAX=SHIFT(MASK(MANTSA),MANTSA)                                  000400
      PMAX=IPMAX*BIG                                                    000410
      PMAX=PMAX.OR.(SHIFT(MASK(MANTLEN),MANTLEN))                       000420
      PMAY=SHIFT(SHIFT(PMAX,-KEY(7)),KEY(7))                            000430
C        SHIFTING PMAX DISABLES THE MODE CONVERSION TO INTEGER FOR KEY(2)000440
      KEY(2)=SHIFT(PMAX,0)                                              000450
      MAXM=SHIFT(MASK(1),MANTSA)                                        000460
      PRR=MAXM                                                          000470
      PMIN=PRR*2.**(-IRNGE)                                             000480
      IF(IPTPOS.EQ.0)    PMIN=PRR*2.**(-IRNGE-MANTSA)                   000490
      KEY(3)=SHIFT(PMIN,0)                                              000500
   23 RETURN                                                            000510
C     SPECIAL CASE FOR 60 BIT WORD MAX,ITS REAL MAX WOULD BOMB THE JOB  000520
C       WHEN ANY COMPARES WERE MADE, SO ITS MADE SLIGHTLY LESS          000530
   25 CONTINUE                                                          000540
      KEY(2)=377777777777777777779                                      000550
      KEY(3)=000140000000000000000                                      000560
      GO TO 23                                                          000570
   30 PRINT *," TOO MANY BITS SPECIFIED; N-BIT MAX IS 60 ",             000580
     *       " --- ",NBITS," WERE SPECIFIED"                            000590
      GO TO 40                                                          000600
   31 PRINT *," MANTISSA + EXPONENT + 1 SHOULD EQUAL 'N'-BITS"          000610
      GO TO 40                                                          000620
   32 PRINT *," ILLEGAL ROUNDING/TRUNCATION OPTION VALUE ",             000630
     *       " ROUNDING = 1/ TRUNCATION = 0"                            000640
      GO TO 40                                                          000650
   33 PRINT *," ILLEGAL AMOUNT OF PRECISION SPECIFIED "                 000660
      PRINT *," ( 1=SINGLE PRECISION, 2=DOUBLE, 3=TRIPLE)"              000670
      GO TO 40                                                          000680
   34 PRINT *," *** NUMBER OF BITS SPECIFIED FOR (MANTISSA*PRECISION)"  000690
     *       " EXCEEDS THAN MAXIMUM PROGRAM CAN HANDLE"                 000700
```

```
      GO TO 40                                                          000710
 35   PRINT *," EXPONENT TOO LARGE FOR CDC-- > 11 BITS"                 000720
 40   PRINT *," ERROR- ILLEGAL SETNBIT DATA PARAMETER FOUND"            000730
      STOP " BAD INPUT TO SETNBIT ROUTINE"                              000740
      END                                                               000750
      INTEGER FUNCTION IIADD(I1,I2,ROUTINE,LNCNT,IFNL,KEY)              000760
      DIMENSION KEY(7)                                                  000770
      IIADD=I1+I2                                                       000780
      IF(IABS(IIADD).GT.KEY(1))   GO TO 100                             000790
 5    RETURN                                                            000800
 100  IF(KEY(5).NE.0)      CALL OVERFLW(ROUTINE,LNCNT,8HADDITION)       000810
      IF(IIADD.GT.0)    IIADD=KEY(1)                                    000820
      IF(IIADD.LT.0)    IIADD=-KEY(1)                                   000830
      GO TO 5                                                           000840
      END                                                               000850
      SUBROUTINE OVERFLW(ROUTINE,LNCNT,OPER)                            000860
      IMPLICIT INTEGER (A-Z)                                            000870
C  ********  MAY WANT TO CHANGE THIS PRINT FROM A FORMATTED ********     000880
      PRINT(1,4) ROUTINE,LNCNT,OPER                                     000890
 4    FORMAT(" **** OVERFLOW IN ",A8," LINE # ",I5," FOR ",A10," ***")  000900
      RETURN                                                            000910
      END                                                               000920
      SUBROUTINE UNDERFLW(ROUTINE,LNCNT,OPER)                           000930
      IMPLICIT INTEGER (A-Z)                                            000940
      PRINT (1,4) ROUTINE,LNCNT,OPER                                    000950
 4    FORMAT(" **** UNDERFLOW IN ",A8," LINE #",I5," FOR ",A10," ***")  000960
      RETURN                                                            000970
      END                                                               000980
      REAL FUNCTION RRADD(R1,R2,ROUTINE,LNCNT,IFNL,KEY)                 000990
      DIMENSION KEY(7)                                                  001000
      RRADD=R1+R2                                                       001010
      IF(KEY(4).NE.0)     GO TO 30                                      001020
 26   IF(ABS(RRADD).GT.SHIFT(KEY(2),0))   GO TO 100                     001030
      IF(ABS(RRADD).LT.SHIFT(KEY(3),0).A.RRADD.NE.0)   GO TO 110        001040
C     CHECK FOR ROUNDING                                                001050
 28   RRADD=SHIFT(SHIFT(RRADD,-KEY(7)),KEY(7))                          001060
      IF(IFNL.EQ.1)  RRADD=SHIFT(SHIFT(RRADD,-KEY(6)),KEY(5))           001070
      RETURN                                                            001080
C     GO ROUND                                                          001090
 30   INC=1                                                             001100
      IF(RRADD.LT.0.)     INC=-1                                        001110
      RRADD=SHIFT(SHIFT(SHIFT(RRADD,-(KEY(7)-1))+INC,-1),KEY(7))        001120
      IF(SHIFT(SHIFT(RRADD,12),-12).EQ.0)   CALL ROUNDER(RRADD)         001130
      GO TO 26                                                          001140
 100  IF(RRADD.GT.0.)      RRADD=SHIFT(KEY(2),0)                        001150
      IF(RRADD.LT.0.)       RRADD=-SHIFT(KEY(2),0)                      001160
C   CHECK FOR MESSAGES                                                  001170
      IF(KEY(5).NE.0)       CALL OVERFLW(ROUTINE,LNCNT,8HADDITION)      001180
      GO TO 28                                                          001190
 110  RRADD=0                                                           001200
      IF(KEY(5).NE.0)       CALL UNDERFLW(ROUTINE,LNCNT,8HADDITION)     001210
      GO TO 28                                                          001220
      END                                                               001230
      REAL FUNCTION R2ADD(R1,R2,ROUTINE,LNCNT,IFNL,KEY)                 001240
      DIMENSION KEY(7)                                                  001250
      R2ADD=R1+R2                                                       001260
      IF(KEY(4).NE.0)     GO TO 30                                      001270
 26   IF(ABS(R2ADD).GT.SHIFT(KEY(2),0))     GO TO 100                   001280
      IF(ABS(R2ADD).LT.SHIFT(KEY(3),0).A.R2ADD.NE.0)   GO TO 110        001290
 28   R2ADD=SHIFT(SHIFT(R2ADD,-KEY(7)),KEY(7))                          001300
      IF(IFNL.EQ.1)     R2ADD=SHIFT(SHIFT(R2ADD,-KEY(6)),KEY(5))        001310
      RETURN                                                            001320
C     GO ROUND IT NOW                                                   001330
 30   INC=1                                                             001340
      IF(R2ADD.LT.0.)     INC=-1                                        001350
      R2ADD=SHIFT(SHIFT(SHIFT(R2ADD,-(KEY(7)-1))+INC,-1),KEY(7))        001360
```

```
      IF(SHIFT(SHIFT(R2ADD,12),-12).EQ.0)   CALL ROUNDER(R2ADD)          001370
      GO TO 25                                                           001380
  100 IF(R2ADD.GT.0.)      R2ADD=SHIFT(KEY(2),0)                         001390
      IF(R2ADD.LT.0.)       R2ADD=-SHIFT(KEY(2),0)                       001400
      IF(KEY(5).NE.0)        CALL OVERFLW(ROUTINE,LNCNT,8HADDITION)      001410
      GO TO 28                                                           001420
  110 R2ADD=0                                                            001430
      IF(KEY(5).NE.0)        CALL UNDRFLW(ROUTINE,LNCNT,8HADDITION)      001440
      GO TO 28                                                           001450
      END                                                                001460
      REAL FUNCTION R1ADD(I1,R2,ROUTINE,LNCNT,IFNL,KEY)                  001470
      DIMENSION KEY(7)                                                   001480
      R1ADD=I1+R2                                                        001490
      IF(KEY(4).NE.0)     GO TO 30                                       001500
   26 IF(ABS(R1ADD).GT.SHIFT(KEY(2),0))     GO TO 100                    001510
      IF (ABS(R1ADD).LT.SHIFT(KEY(3),0).A.R1ADD.NE.0)    GO TO 110       001520
   28 R1ADD=SHIFT(SHIFT(R1ADD,-KEY(7)),KEY(7))                           001530
      IF(IFNL.EQ.1)   R1ADD=SHIFT(SHIFT(R1ADD,-KEY(6)),KEY(6))           001540
      RETURN                                                             001550
C     GO ROUND IT NOW                                                    001560
   30 INC=1                                                              001570
      IF(R1ADD.LT.0.)     INC=-1                                         001580
      R1ADD=SHIFT(SHIFT(SHIFT(R1ADD,-(KEY(7)-1))+INC,-1),KEY(7))         001590
      IF(SHIFT(SHIFT(R1ADD,12),-12).EQ.0)  CALL ROUNDER(R1ADD)           001600
      GO TO 25                                                           001610
  100 IF(RRADD.GT.0.)     RRADD=SHIFT(KEY(2),0)                          001620
      IF(RRADD.LT.0.)     RRADD=-SHIFT(KEY(2),0)                         001630
      IF(KEY(5).NE.0)     CALL OVERFLW(ROUTINE,LNCNT,8HADDITION)         001640
      GO TO 28                                                           001650
  110 R1ADD=0                                                            001660
      IF(KEY(5).NE.0)     CALL UNDRFLW(ROUTINE,LNCNT,8HADDITION)         001670
      GO TO 28                                                           001680
      END                                                                001690
      INTEGER FUNCTION IIMPY(I1,I2,ROUTINE,LNCNT,IFNL,KEY)               001700
      DIMENSION KEY(7)                                                   001710
      IIMPY=I1*I2                                                        001720
      IF(IABS(IIMPY).GT.KEY(1))   GO TO 100                              001730
    5 RETURN                                                             001740
  100    CALL OVERFLW(ROUTINE,LNCNT,8HMULTIPLY)                          001750
      IF(IIMPY.GT.0)     IIMPY=KEY(1)                                    001760
      IF(IIMPY.LT.0)     IIMPY=-KEY(1)                                   001770
      GO TO 5                                                            001780
      END                                                                001790
      INTEGER FUNCTION IIMNS(I1,I2,ROUTINE,LNCNT,IFNL,KEY)               001800
      DIMENSION KEY(7)                                                   001810
      IIMNS=I1-I2                                                        001820
      IF(IABS(IIMNS).GT.KEY(1))    GO TO 100                             001830
    5 RETURN                                                             001840
  100 IF(KEY(5).NE.0)     CALL OVERFLW(ROUTINE,LNCNT,8HSUBTRACT)         001850
      IF(IIMNS.GT.0)    IIMNS=KEY(1)                                     001860
      IF(IIMNS.LT.0)      IIMNS=-KEY(1)                                  001870
      GO TO 5                                                            001880
      END                                                                001890
      INTEGER FUNCTION IIDVD(I1,I2,ROUTINE,LNCNT,IFNL,KEY)               001900
      DIMENSION KEY(7)                                                   001910
      IIDVD=I1/I2                                                        001920
      IF(IABS(IIDVD).GT.KEY(1))   GO TO 100                              001930
    5 RETURN                                                             001940
  110 IF(KEY(5).NE.0)    CALL OVERFLW(ROUTINE,LNCNT,8HDIVISION)          001950
      IF(IIDVD.GT.0)    IIDVD=KEY(1)                                     001960
      IF(IIDVD.LT.0)    IIDVD=-KEY(1)                                    001970
      GO TO 5                                                            001980
      END                                                                001990
      INTEGER FUNCTION IIEXP(I1,I2,ROUTINE,LNCNT,IFNL,KEY)               002000
      DIMENSION KEY(7)                                                   002010
      IIEXP=I1**I2                                                       002020
```

```
      IF(IABS(IIEXP).GT.KEY(1))  GO TO 100                              002030
    5 RETURN                                                            002040
  100 IF(KEY(5).NE.0)     CALL OVERFLW(ROUTINE,LNCNT,8HEXPONENT)        002050
      IF(IIEXP.GT.0)     IIEXP=KEY(1)                                   002060
      IF(IIEXP.LT.0)    IIEXP=-KEY(1)                                   002070
      GO TO 5                                                           002080
      END                                                               002090
      REAL FUNCTION RPMPY(R1,P2,ROUTINE,LNCNT,IFNL,KEY)                 002100
      DIMENSION KEY(7)                                                  002110
      RPMPY=R1*P2                                                       002120
      IF(KEY(4).NE.0)     GO TO 30                                      002130
   25 IF(ABS(RPMPY).GT.SHIFT(KEY(2),0))    GO TO 100                    002140
      IF(ABS(RPMPY).LT.SHIFT(KEY(3),0).A.RPMPY.NE.0)    GO TO 110       002150
   28 RPMPY=SHIFT(SHIFT(RPMPY,-KEY(7)),KEY(7))                          002160
      IF(IFNL.EQ.1)  RPMPY=SHIFT(SHIFT(RPMPY,-KEY(5)),KEY(5))           002170
      RETURN                                                            002180
C     GO ROUND                                                          002190
   30 INC=1                                                             002200
      IF(RPMPY.LT.0.)     INC=-1                                        002210
      RPMPY=SHIFT(SHIFT(SHIFT(RPMPY,-(KEY(7)-1))+INC,-1),KEY(7))        002220
      IF(SHIFT(SHIFT(RPMPY,12),-12).EQ.0)       CALL ROUNDER(RPMPY)     002230
      GO TO 25                                                          002240
  100 IF(RPMPY.GT.0.)       RPMPY=SHIFT(KEY(2),0)                       002250
      IF(RPMPY.LT.0.)     RPMPY=-SHIFT(KEY(2),0)                        002260
C     CHECK FOR MESSAGES                                                002270
      IF(KEY(5).NE.0)     CALL OVERFLW(ROUTINE,LNCNT,8HMULTIPLY)        002280
      GO TO 28                                                          002290
  110 RPMPY=0                                                           002300
      IF(KEY(5).NE.0)     CALL UNDRFLW(ROUTINE,LNCNT,8HMULTIPLY)        002310
      GO TO 28                                                          002320
      END                                                               002330
      REAL FUNCTION RPMNS(R1,P2,ROUTINE,LNCNT,IFNL,KEY)                 002340
      DIMENSION KEY(7)                                                  002350
      RPMNS=R1-P2                                                       002360
      IF(KEY(4).NE.0)      GO TO 30                                     002370
   25 IF(ABS(RPMNS).GT.SHIFT(KEY(2),0))     GO TO 100                   002380
      IF(ABS(RPMNS).LT.SHIFT(KEY(3),0).A.RPMNS.NE.0)    GO TO 110       002390
   28 RPMNS=SHIFT(SHIFT(RPMNS,-KEY(7)),KEY(7))                          002400
      IF(IFNL.EQ.1)  RPMNS=SHIFT (SHIFT(RPMNS,-KEY(5)),KEY(5))          002410
      RETURN                                                            002420
C     GO ROUND                                                          002430
   30 INC=1                                                             002440
      IF(RPMNS.LT.0.)     INC=-1                                        002450
      RPMNS=SHIFT(SHIFT(SHIFT(RPMNS,-(KEY(7)-1))+INC,-1),KEY(7))        002460
      IF(SHIFT(SHIFT(RPMNS,12),-12).EQ.0)   CALL  ROUNDER(RPMNS)        002470
      GO TO 25                                                          002480
  100 IF(RPMNS.GT.0.)     RPMNS=SHIFT(KEY(2),0)                         002490
      IF(RPMNS.LT.0.)      RPMNS=-SHIFT(KEY(2),0)                       002500
C     CHECK FOR MESSAGES                                                002510
      IF(KEY(5).NE.0)     CALL OVERFLW(ROUTINE,LNCNT,8HSUBTRACT)        002520
      GO TO 28                                                          002530
  110 RPMNS=0                                                           002540
      IF(KEY(5).NE.0)       CALL UNDRFLW(ROUTINE,LNCNT,8HSUBTRACT)      002550
      GO TO 28                                                          002560
      END                                                               002570
      REAL FUNCTION RPDVD(R1,P2,ROUTINE,LNCNT,IFNL,KEY)                 002580
      DIMENSION KEY(7)                                                  002590
      RPDVD=R1/P2                                                       002600
      IF(KEY(4).NE.0)     GO TO 30                                      002610
   25 IF(ABS(RPDVD).GT.SHIFT(KEY(2),0))     GO TO 100                   002620
      IF(ABS(RPDVD).LT.SHIFT(KEY(3),0).A.RPDVD.NE.0)    GO TO 110       002630
   28 RPDVD=SHIFT(SHIFT(RPDVD,-KEY(7)),KEY(7))                          002640
      IF(IFNL.EQ.1)  RPDVD=SHIFT(SHIFT(RPDVD,-KEY(5)),KEY(5))           002650
      RETURN                                                            002660
C     GO ROUND                                                          002670
   30 INC=1                                                             002680
```

```
 30   INC=1                                                              002690
      IF(RRDVD.LT.0.)      INC=-1                                        002700
      RRDVD=SHIFT(SHIFT(SHIFT(RRDVD,-(KEY(7)-1))+INC,-1),KEY(7))         002710
      IF(SHIFT(SHIFT(RRDVD,12),-12).EQ.0)   CALL ROUNDER(RRDVD)          002720
      GO TO 26                                                           002730
 100  IF(RRDVD.GT.0.)      RRDVD=SHIFT(KEY(2),0)                         002740
      IF(RRDVD.LT.0.)      RRDVD=-SHIFT(KEY(2),0)                        002750
C     CHECK FOR MESSAGES                                                 002760
      IF(KEY(5).NE.0)      CALL OVERFLW(ROUTINE,LNCNT,8HDIVISION)        002770
      GO TO 28                                                           002780
 110  RRDVD=0                                                            002790
      IF(KEY(5).NE.0)        CALL UNDRFLW(ROUTINE,LNCNT,8HDIVISION)      002800
      GO TO 28                                                           002810
      END                                                                002820
      REAL FUNCTION R2MNS(R1,I2,ROUTINE,LNCNT,IFNL,KEY)                  002830
      DIMENSION KEY(7)                                                   002840
      R2MNS=R1-I2                                                        002850
C      CHECK FOR ROUNDING                                                002860
      IF(KEY(4).NE.0)    GO TO 30                                        002870
 26   IF(ABS(R2MNS).GT.SHIFT(KEY(2),0))     GO TO 100                    002880
      IF(ABS(R2MNS).LT.SHIFT(KEY(3),0).A.R2MNS.NE.0)   GO TO 110         002890
 28   R2MNS=SHIFT(SHIFT(R2MNS,-KEY(7)),KEY(7))                           002900
      IF(IFNL.EQ.1)     R2MNS=SHIFT(SHIFT(R2MNS,-KEY(6)),KEY(6))         002910
      RETURN                                                             002920
C     GO ROUND IT NOW                                                    002930
 30   INC=1                                                              002940
      IF(R2MNS.LT.0.)       INC=-1                                       002950
      R2MNS=SHIFT(SHIFT(SHIFT(R2MNS,-(KEY(7)-1))+INC,-1),KEY(7))         002960
      IF(SHIFT(SHIFT(R2MNS,12),-12).EQ.0)    CALL    ROUNDER(R2MNS)      002970
      GO TO 26                                                           002980
 100  IF(R2MNS.GT.0.)      R2MNS=SHIFT(KEY(2),0)                        002990
      IF(R2MNS.LT.0.)      R2MNS=-SHIFT(KEY(2),0)                        003000
      IF(KEY(5).NE.0)        CALL OVERFLW(ROUTINE,LNCNT,8HSUBTRACT)      003010
      GO TO 28                                                           003020
 110  R2MNS=0                                                            003030
      IF(KEY(5).NE.0)        CALL UNDRFLW(ROUTINE,LNCNT,8HSUBTRACT)      003040
      GO TO 28                                                           003050
      END                                                                003060
      REAL FUNCTION R1MNS(I1,R2,ROUTINE,LNCNT,IFNL,KEY)                  003070
      DIMENSION KEY(7)                                                   003080
      R1MNS=I1-R2                                                        003090
C       CHECK FOR ROUNDING OPTION                                        003100
      IF(KEY(4).NE.0)     GO TO 30                                       003110
 26   IF(ABS(R1MNS).GT.SHIFT(KEY(2),0))      GO TO 100                   003120
      IF (ABS(R1MNS).LT.SHIFT(KEY(3),0).A.R1MNS.NE.0)    GO TO 110       003130
 28   R1MNS=SHIFT(SHIFT(R1MNS,-KEY(7)),KEY(7))                           003140
      IF(IFNL.EQ.1)   R1MNS=SHIFT(SHIFT(R1MNS,-KEY(6)),KEY(6))           003150
      RETURN                                                             003160
C     GO ROUND IT NOW                                                    003170
 30   INC=1                                                              003180
      IF(R1MNS.LT.0)  INC=-1                                             003190
      R1MNS=SHIFT(SHIFT(SHIFT(R1MNS,-(KEY(7)-1))+INC,-1),KEY(7))         003200
      IF(SHIFT(SHIFT(R1MNS,12),-12).EQ.0)  CALL ROUNDER(R1MNS)           003210
      GO TO 26                                                           003220
 100  IF(R1MNS.GT.0.)    R1MNS=SHIFT(KEY(2),0)                          003230
      IF(R1MNS.LT.0.)     R1MNS=-SHIFT(KEY(2),0)                         003240
      IF(KEY(5).NE.0)     CALL OVERFLW(ROUTINE,LNCNT,8HSUBTRACT)         003250
      GO TO 28                                                           003260
 110  R1MNS=0                                                            003270
      IF(KEY(5).NE.0)       CALL UNDRFLW(ROUTINE,LNCNT,8HSUBTRACT)       003280
      GO TO 28                                                           003290
      END                                                                003300
      REAL FUNCTION R2MPY(R1,I2,ROUTINE,LNCNT,IFNL,KEY)                  003310
      DIMENSION KEY(7)                                                   003320
      R2MPY=R1*I2                                                        003330
C       CHECK FOR ROUNDING                                               003340
      IF(KEY(4).NE.0)     GO TO 30                                       003350
```

```
      IF(KEY(4).NE.0)      GO TO 30
 25   IF(ABS(R2MPY).GT.SHIFT(KEY(2),0))      GO TO 100
      IF(ABS(R2MPY).LT.SHIFT(KEY(3),0).A.R2MPY.NE.0)    GO TO 110
 28   R2MPY=SHIFT(SHIFT(R2MPY,-KEY(7)),KEY(7))
      IF(IFNL.EQ.1)     R2MPY=SHIFT(SHIFT(R2MPY,-KEY(6)),KEY(5))
      RETURN
C     GO ROUND IT NOW
 30   INC=1
      IF(R2MPY.LT.0.)      INC=-1
      R2MPY=SHIFT(SHIFT(SHIFT(R2MPY,-(KEY(7)-1))+INC,-1),KEY(7))
      IF(SHIFT(SHIFT(R2MPY,12),-12).EQ.0)   CALL ROUNDER(R2MPY)
      GO TO 25
 100  IF(R2MPY.GT.0.)      R2MPY=SHIFT(KEY(2),0)
      IF(R2MPY.LT.0.)     R2MPY=-SHIFT(KEY(2),0)
      IF(KEY(5).NE.0)    CALL OVERFLW(ROUTINE,LNCNT,8HMULTIPLY)
      GO TO 28
 110  R2MPY=0
      IF(KEY(5).NE.0)      CALL UNDRFLW(ROUTINE,LNCNT,8HMULTIPLY)
      GO TO 28
      END
      REAL FUNCTION R1MPY(I1,R2,ROUTINE,LNCNT,IFNL,KEY)
      DIMENSION KEY(7)
      R1MPY=I1*R2
C       CHECK FOR ROUNDING
      IF(KEY(4).NE.0)      GO TO 30
 25   IF(ABS(R1MPY).GT.SHIFT(KEY(2),0))       GO TO 100
      IF (ABS(R1MPY).LT.SHIFT(KEY(3),0).A.R1MPY.NE.0)     GO TO 110
 28   R1MPY=SHIFT(SHIFT(R1MPY,-KEY(7)),KEY(7))
      IF(IFNL.EQ.1)    R1MPY=SHIFT(SHIFT(R1MPY,-KEY(6)),KEY(5))
      RETURN
C     GO ROUND IT NOW
 30   INC=1
      IF(R1MPY.LT.0.)       INC=-1
      R1MPY=SHIFT(SHIFT(SHIFT(R1MPY,-(KEY(7)-1))+INC,-1),KEY(7))
      IF(SHIFT(SHIFT(R1MPY,12),-12).EQ.0)   CALL   ROUNDER(R1MPY)
      GO TO 25
 100  IF(R1MPY.GT.0.)     R1MPY=SHIFT(KEY(2),0)
      IF(R1MPY.LT.0.)     R1MPY=-SHIFT(KEY(2),0)
      IF (KEY(5).NE.0)  CALL OVERFLW(ROUTINE,LNCNT,8HMULTIPLY)
      GO TO 28
 110  R1MPY=0
      IF(KEY(5).NE.0)    CALL UNDRFLW(ROUTINE,LNCNT,8HMULTIPLY)
      GO TO 28
      END
      REAL FUNCTION R2DVD(R1,I2,ROUTINE,LNCNT,IFNL,KEY)
      DIMENSION KEY(7)
      R2DVD=R1/I2
C       CHECK FOR ROUNDING
      IF(KEY(4).NE.0)      GO TO 30
 25   IF(ABS(R2DVD).GT.SHIFT(KEY(2),0))      GO TO 100
      IF(ABS(R2DVD).LT.SHIFT(KEY(3),0).A.R2DVD.NE.0)    GO TO 110
 28   R2DVD=SHIFT(SHIFT(R2DVD,-KEY(7)),KEY(7))
      IF(IFNL.EQ.1)     R2DVD=SHIFT(SHIFT(R2DVD,-KEY(6)),KEY(5))
      RETURN
C     GO ROUND IT NOW
 30   INC=1
      IF(R2DVD.LT.0.)      INC=-1
      R2DVD=SHIFT(SHIFT(SHIFT(R2DVD,-(KEY(7)-1))+INC,-1),KEY(7))
      IF(SHIFT(SHIFT(R2DVD,12),-12).EQ.0)    CALL ROUNDER(R2DVD)
      GO TO 25
 100  IF(R2DVD.GT.0.)      R2DVD=SHIFT(KEY(2),0)
      IF(R2DVD.LT.0.)       R2DVD=-SHIFT(KEY(2),0)
      IF(KEY(5).NE.0)          CALL OVERFLW(ROUTINE,LNCNT,8HDIVISION)
      GO TO 28
 110  R2DVD=0
      IF(KEY(5).NE.0)          CALL UNDRFLW(ROUTINE,LNCNT,8HDIVISION)
      GO TO 28
```

```
      GO TO 20                                                          004000
      END                                                               004010
      REAL FUNCTION R1DVD(I1,R2,ROUTINE,LNCNT,IFNL,KEY)                 004020
      DIMENSION KEY(7)                                                  004030
      R1DVD=I1/R2                                                       004040
C      CHECK FOR ROUNDING                                               004050
      IF(KEY(4).NE.0)      GO TO 30                                     004060
 26   IF(ABS(R1DVD).GT.SHIFT(KEY(2),0))      GO TO 100                  004070
      IF (ABS(R1DVD).LT.SHIFT(KEY(3),0).A.R1DVD.NE.0)     GO TO 110     004080
 28   R1DVD=SHIFT(SHIFT(R1DVD,-KEY(7)),KEY(7))                          004090
      IF(IFNL.EQ.1)    R1DVD=SHIFT(SHIFT(R1DVD,-KEY(6)),KEY(6))         004100
      RETURN                                                            004110
C     GO ROUND IT NOW                                                   004120
 30   INC=1                                                             004130
      IF(R1DVD.LT.0.)      INC=-1                                       004140
      R1DVD=SHIFT(SHIFT(SHIFT(R1DVD,-(KEY(7)-1))+INC,-1),KEY(7))        004150
      IF(SHIFT(SHIFT(R1DVD,12),-12).EQ.0)  CALL ROUNDER(R1DVD)          004160
      GO TO 25                                                          004170
 100  IF(R1DVD.GT.0.)     R1DVD=SHIFT(KEY(2),0)                        004180
      IF(R1DVD.LT.0.)    R1DVD=-SHIFT(KEY(2),0)                        004190
      IF(KEY(5).NE.0)    CALL OVERFLW(ROUTINE,LNCNT,8HDIVISION)         004200
      GO TO 28                                                          004210
 110  R1DVD=0                                                           004220
      IF(KEY(5).NE.0)    CALL UNDRFLW(ROUTINE,LNCNT,8HDIVISION)         004230
      GO TO 28                                                          004240
      END                                                               004250
      REAL FUNCTION R2EXP(R1,I2,ROUTINE,LNCNT,IFNL,KEY)                 004260
      DIMENSION KEY(7)                                                  004270
      R2EXP=R1**I2                                                      004280
      IF(KEY(4).NE.0)     GO TO 30                                      004290
 26   IF(ABS(R2EXP).GT.SHIFT(KEY(2),0))     GO TO 100                   004300
      IF(ABS(R2EXP).LT.SHIFT(KEY(3),0).A.R2EXP.NE.0)   GO TO 110        004310
 28   R2EXP=SHIFT(SHIFT(R2EXP,-KEY(7)),KEY(7))                          004320
      IF(IFNL.EQ.1)     R2EXP=SHIFT(SHIFT(R2EXP,-KEY(6)),KEY(6))        004330
      RETURN                                                            004340
C     GO ROUND IT NOW                                                   004350
 30   INC=1                                                             004360
      IF(R2EXP.LT.0.)      INC=-1                                       004370
      R2EXP=SHIFT(SHIFT(SHIFT(R2EXP,-(KEY(7)-1))+INC,-1),KEY(7))        004380
      IF(SHIFT(SHIFT(R2EXP,12),-12).EQ.0)   CALL ROUNDER(R2EXP)         004390
      GO TO 28                                                          004400
 100  IF(R2EXP.GT.0.)      R2EXP=SHIFT(KEY(2),0)                       004410
      IF(R2EXP.LT.0.)      R2EXP=-SHIFT(KEY(2),0)                      004420
      IF(KEY(5).NE.0)        CALL OVERFLW(ROUTINE,LNCNT,8HEXPONENT)     004430
      GO TO 28                                                          004440
 110  R2EXP=0                                                           004450
      IF(KEY(5).NE.0)        CALL UNDRFLW(ROUTINE,LNCNT,8HEXPONENT)     004460
      GO TO 28                                                          004470
      END                                                               004480
      FUNCTION IASGN(I1,ROUTINE,LNCNT,IFNL,KEY)                         004490
      DIMENSION KEY(7)                                                  004500
      IASGN=I1                                                          004510
      IF(I1.GT.KEY(1))    IASGN=KEY(1)                                  004520
      IF(I1.LT.-KEY(1))   IASGN=-KEY(1)                                 004530
      RETURN                                                            004540
      END                                                               004550
      FUNCTION RASGN(R1,ROUTINE,LNCNT,IFNL,KEY)                         004560
      DIMENSION KEY(7)                                                  004570
      RR1=R1                                                            004580
      IF(RR1.GT.SHIFT(KEY(2),0))    RR1=SHIFT(KEY(2),0)                 004590
      IF(RR1.LT.-SHIFT(KEY(2),0))   RR1=-SHIFT(KEY(2),0)                004600
      IF(ABS(RR1).LT.SHIFT(KEY(3),0).A.RR1.NE.0) RR1=0                  004610
      RASGN=SHIFT(SHIFT(RR1,-KEY(7)),KEY(7))                            004620
      RETURN                                                            004630
      END                                                               004640
      SUBROUTINE ROUNDER(R1,R2,ROUTINE,LNCNT,IFNL,KEY)                  004650
```

```
      DIMENSION KEY(7)
      RREXP=R1**R2
      IF(KEY(4).NE.0)     GO TO 30
25    IF(ABS(RREXP).GT.SHIFT(KEY(2),0))     GO TO 100
      IF(ABS(RREXP).LT.SHIFT(KEY(3),0).A.RREXP.NE.0)     GO TO 110
28    RREXP=SHIFT(SHIFT(RREXP,-KEY(7)),KEY(7))
      IF(IFNL.EQ.1)    RREXP=SHIFT(SHIFT(RREXP,-KEY(6)),KEY(6))
      RETURN
30    INC=1
      IF(RREXP.LT.0.)     INC=-1
      RREXP=SHIFT(SHIFT(SHIFT(RREXP,-(KEY(7)-1))+INC,-1),KEY(7))
      IF(SHIFT(SHIFT(RREXP,12),-12).EQ.0)   CALL ROUNDER(RREXP)
      GO TO 25
100   IF(RREXP.GT.0.)     RREXP=SHIFT(KEY(2),0)
      IF(RREXP.LT.0.)     RREXP=-SHIFT(KEY(2),0)
      IF(KEY(5).NE.0)     CALL OVERFLW(ROUTINE,LNCNT,8HEXPONENT)
      GO TO 28
110   RREXP=0
      IF(KEY(5).NE.0)     CALL UNDRFLW(ROUTINE,LNCNT,8HEXPONENT)
      GO TO 28
      END
      REAL FUNCTION R1EXP(I1,R2,ROUTINE,LNCNT,IFNL,KEY)
      DIMENSION KEY(7)
      R1EXP=I1**R2
      IF(KEY(4).NE.0)     GO TO 30
25    IF(ABS(R1EXP).GT.SHIFT(KEY(2),0))     GO TO 100
      IF(ABS(R1EXP).LT.SHIFT(KEY(3),0).A.R1EXP.NE.0)     GO TO 110
28    R1EXP=SHIFT(SHIFT(R1EXP,-KEY(7)),KEY(7))
      IF(IFNL.EQ.1)     R1EXP=SHIFT(SHIFT(R1EXP,-KEY(6)),KEY(6))
      RETURN
30    INC=1
      IF(R1EXP.LT.0.)    INC=-1
      R1EXP=SHIFT(SHIFT(SHIFT(R1EXP,-(KEY(7)-1))+INC,-1),KEY(7))
      IF(SHIFT(SHIFT(R1EXP,12),-12).EQ.0)      CALL ROUNDER(R1EXP)
      GO TO 25
100   IF(R1EXP.GT.0.)     R1EXP=SHIFT(KEY(2),0)
      IF(R1EXP.LT.0.)    R1EXP=-SHIFT(KEY(2),0)
      IF(KEY(5).NE.0)    CALL OVERFLW(ROUTINE,LNCNT,8HEXPONENT)
      GO TO 28
110   R1EXP=0
      IF(KEY(5).NE.0)     CALL UNDRFLW(ROUTINE,LNCNT,8HEXPONENT)
      GO TO 28
      END
      SUBROUTINE ROUNDER(RNUM)
      DATA  ONE/00004000000000000000B/,ZERO/77773777777777777777B/
      DATA R1777/17770000000000000000B/
      DATA R6000/60007777777777777777B/
      IF(SHIFT(RNUM,0).EQ.SHIFT(R1777,0)) RNUM=20004000000000000000B
      IF(SHIFT(RNUM,0).EQ.SHIFT(R6000,0)) RNUM=57773777777777777777B
      IF(SHIFT(RNUM,1).LT.0)      RNUM=RNUM.AND.ZERO
      IF(SHIFT(RNUM,1).GT.0)    RNUM=RNUM.OR.ONE
      RETURN
      END
```

Appendix D

Proposed In-line Modification

A proposed change to the existing n-bit simulation tool would replace all subroutine calls generated by the present preprocessor program with in-line code which would perform the n-bit wordlength effects. Although this modification would result in large memory requirements for the resulting preprocessed program, the execution time should be reduced by one-third. The in-line code would be tailored to the user options in effect for that particular program module. This would mean fewer decisions would be made than in the present simulation tool, during execution of the n-bit simulated version of the algorithm. Also, subroutine linkage time would be save since the n-bit wordlength effects were no longer performed through calls to subroutines.

This modification would require the detection of the CALL SETNBIT subroutine for each program unit. The NON-EXPRESSION-HANDLER module could be modified to detect CALL SETNBIT, then call a handling routine which could be developed to analyze the user options expressed as arguments in SETNBIT. The same SETNBIT subroutine which is used presently could compute the key values and pass them via COMMON to the subroutines of the preprocessor which output the n-bit simulation code for the arithmetic expressions.

The in-line code which could perform the n-bit simulation effects for the various arithmetic operations and operand data types is shown in Figure D-1. The underlined words in the figure would be filled in by the handling routine with each n-bit simulation situation.

```
For Real
 Results:   object = varb1 operation varb2

If Round
 Option
 is on      object = SHIFT(SHIFT(SHIFT(object ,-(key7 -1))+
                     (-1**(SHIFT(object .AND. MASK(1),1)),
                        -1),key7 )

            IF(SHIFT(SHIFT(object ,12),-12).eq.0)
                        CALL ROUNDER(object)

            IF( ABS(object).GT.key2 .OR.(ABS(object).LT.key3
               .AND.ABS(object).NE.0)
                  CALL OVRFLWR(object ,operation  ,routine,
                        line# ,key2 )

            object = SHIFT(SHIFT(object ,-key6/7), key6/7)

For Integer
  Results:  object = varb1 operation varb2
            IF(IABS(object).GT.key1)
                  CALL OVRFLWI(object ,operation ,routine,
                        line#, key1)
```

Figure D-1 Possible In-line Code For
Real and Integer Arithmetic Operation Results

To output the code shown in Figure D-1, the GETPRTS and SNGLASG modules would have to be modified slightly. Specifically, the merge operations of each module would be replaced by the actual arithmetic operation build followed by a subroutine call to a new module. This new module would have standard outputs (shown by the non-underlined portions of Figure D-1). Key values (computed by SETNBIT) would be placed in the standard positions. The calling module would have to pass to this new module the object name, object data type, and the final assignment flag. In addition, the new routine must

be passed values computed by the SETNBIT subroutine. In Figure D-1, if the final flag was turned on, key6 would be the value positioned in the last statement. If this was not the final assignment key7 would be placed there, so a multi-precision mantissa would be saved when the in-line code is executed.

Vita

Gary A. Klein was born on 18 June 1951 in Newton, Iowa, the son of Clifford M. Klein and Doris G. Klein. After graduating from Newton High School in 1969, he attended Iowa State University in Ames, Iowa. In May, 1973 he graduated with distinction from Iowa State with a Bachelor of Science degree in Computer Science and received his commission as a graduate of the ROTC program. One month after graduation, he entered the Air Force as a Second Lieutenant and was stationed at Offutt Air Force Base for three years as a computer programmer. In May, 1976 he entered graduate studies in Electrical Engineering at the Air Force Institute of Technology.

Permanent Address:

Rural Route 1
Newton, Iowa 50208

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFIT/GCS/EE/77-6	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Software Tool(s) For Evaluating The Effects Of Finite Wordlength	5. TYPE OF REPORT & PERIOD COVERED: MS Thesis
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Gary A. Klein, Captain	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Air Force Institute of Technology(AFIT-EN) Wright-Patterson AFB, Ohio 45433	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Avionics Laboratory AFAL/RWA-3 Wright-Patterson AFB, Ohio 45433	12. REPORT DATE: December, 1977
	13. NUMBER OF PAGES: 172
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17
Jerral F. Guess, Captain USAF
Director of Information

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Wordlength	Computer modelling
Computer simulation program	Software performance specifications
Software tool	
Finite-difference computer program	Computerized simulation
Computer wordlength effects	Computer requirements

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In the development of estimation and control systems, the algorithms developed are, in general, very sensitive to the wordlength of the onboard computer. The representation of the desired design can be greatly affected by the quantization resulting from truncation or round off. The result can be even more severe than numerical imprecision: numerical instability can be generated that render algortihms totally useless.

The objective of this thesis was to process algorithms written in FORTRAN on the Control Data Corporation (CDC) 6600/ CYBER 74 computer systems such that the results obtained were similar to those obtainable on an n-bit machine. The problem of n-bit simulation was addressed from two levels: the assembly language level and the FORTRAN language level. Each level involved designing a preprocessor which would modify the algorithm's code so that the numerical effects of n-bit wordlength could be realized. The assembly language (COMPASS) level approach failed, however, an n-bit simulation tool was successfully developed and implemented at the FORTRAN level. Several user options were incorporated into the n-bit simulation tool to enable the user to simulate the characteristics of various computer types.